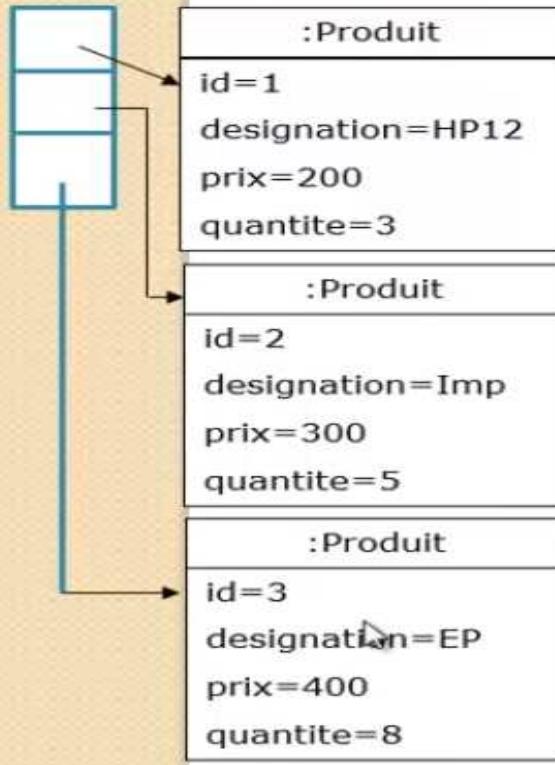


Application Orientée Objet

produits:List



```
public class Produit{  
    private Long id;  
    private String designation;  
    private double prix;  
    private int quantite;
```

Mapping Objet Relationnel

```
public List<Produit> produitsParMC(String mc) {  
    List<Produit> produits=new ArrayList<Produit>();  
    Class.forName("com.mysql.jdbc.Driver");  
    Connection conn=DriverManager.getConnection  
        ("jdbc:mysql://localhost:3306/DB_CAT","root","");
    PreparedStatement ps=conn.prepareStatement("SELECT * FROM  
    PRODUITS WHERE DESIGNATION like ?");  
    ps.setString(1, mc);  
    ResultSet rs=ps.executeQuery();  
    while(rs.next()){  
        Produit p=new Produit();  
        p.setId(rs.getLong("ID"));  
        p.setDesignation(rs.getString("DESIGNATION"));  
        p.setPrix(rs.getDouble("PRIX"));  
        p.setQuantite(rs.getInt("QUANTITE"));  
        produits.add(p);  
    }  
    return produits;  
}
```

SGBDR MySQL, BD DB_CAT

Table PRODUITS

ID	DESIGNATION	PRIX	QUANTITE
1	Ordi HL 3421	980	12
2	Imprimante HP LX 7600	2300	10
3	Imprimante Epson HR 450	1300	10

Introduction

- Travailler dans les deux univers que sont l'orienté objet et la base de données relationnelle peut être lourd et consommateur en temps dans le monde de l'entreprise d'aujourd'hui.
- Hibernate est un outil de mapping objet/relationnel pour le monde Java.
- Le terme mapping objet/relationnel (ORM) décrit la technique consistant à faire le lien entre la représentation objet des données et sa représentation relationnelle basée sur un schéma SQL.

Introduction

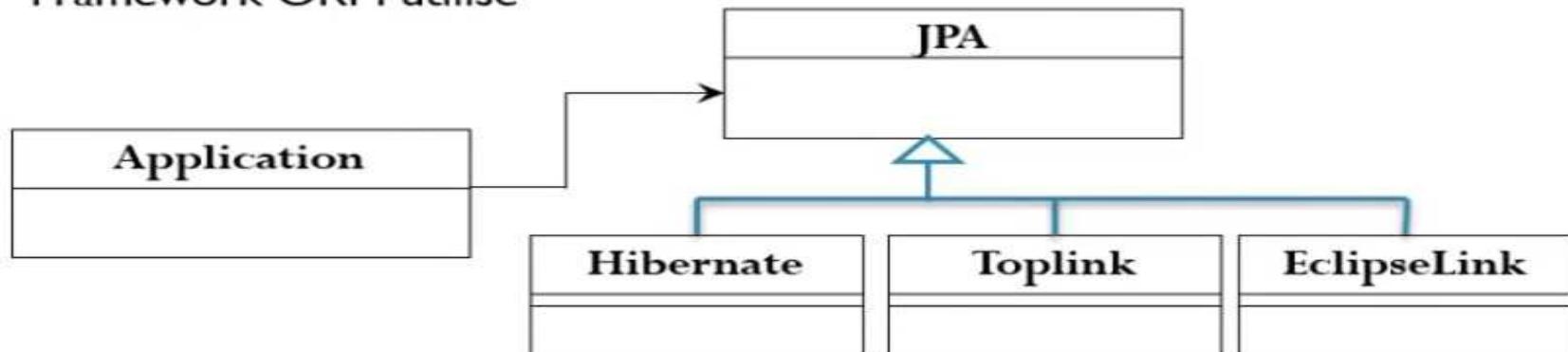
- Hibernate s'occupe du transfert des objets Java dans les tables de la base de données
- En plus, il permet de requêter les données et propose des moyens de les récupérer.
- Il peut donc réduire de manière significative le temps de développement qui aurait été autrement perdu dans une manipulation manuelle des données via SQL et JDBC

But de Hibernate

- Le but d'Hibernate est de libérer le développeur de 95 pourcent des tâches de programmation liées à la persistance des données communes.
- Hibernate assure la portabilité de votre application si vous changer de SGBD.
- Hibernate propose au développeur des méthodes d'accès aux bases de données plus efficace ce qui devrait rassurer les développeurs.
- Maven est utile pour les applications dont la couche métier est implémentée au niveau de l'application et non au niveau du SGBD en utilisant des procédures stockées.

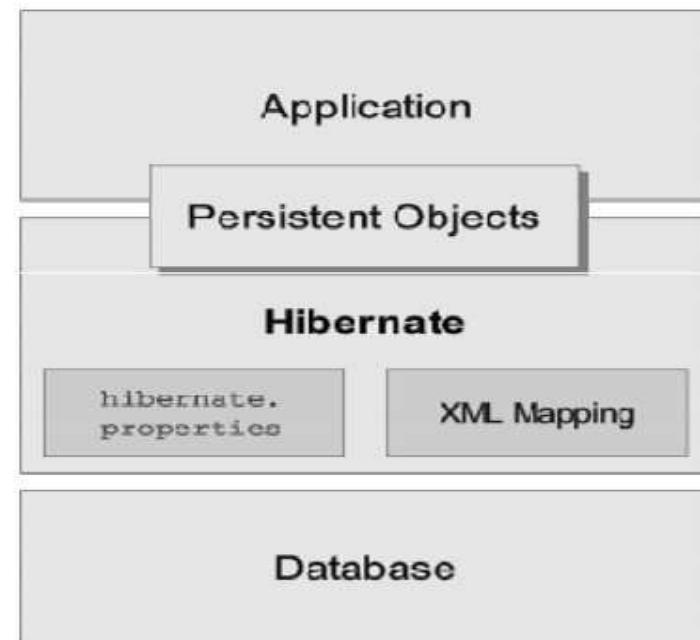
JPA : Java Persistence API

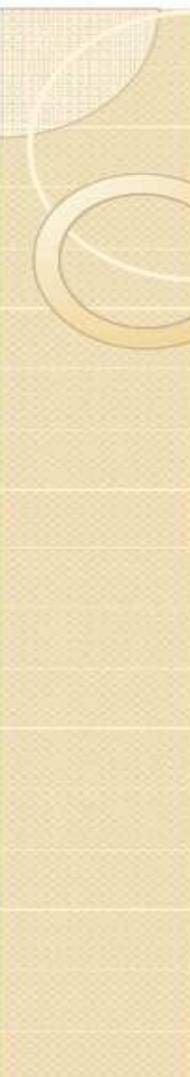
- JPA est une spécification créée par Sun pour standardiser le mapping Objet relationnel.
- JPA définit un ensemble d'interfaces, de classes abstraites et d'annotations qui permettent la description du mapping objet relationnel
- Il existe plusieurs implémentation de JPA:
 - Hibernate
 - Toplink
 - iBatis 
 - EclipseLink
- L'utilisation de JPA permet à votre application d'être indépendante du Framework ORM utilisé



Première approche de l'architecture d'Hibernate

- Hibernate permet d'assurer la persistance des objets de l'application dans un entrepôt de données.
- Cet entrepôt de données est dans la majorité des cas une base de données relationnelle, mais il peut être un fichier XML.
- Le mapping des objets est effectuée par Hibernate en se basant sur des fichiers de configuration en format texte ou souvent XML.





Mapping Objet Relationnel des entités

- Il existe deux moyens pour mapper les entités :
 - Créer des fichier XML de mapping
 - Utiliser les Annotations JPA
- L'utilisation des annotations JPA laisse votre code indépendant de Hibernate.
- La création des fichiers XML de mapping a l'avantage de séparer le code java du mapping objet relationnel.
- Dans cette formation, nous allons utiliser les annotations JPA

Quelques annotations JPA de Mapping des Entités

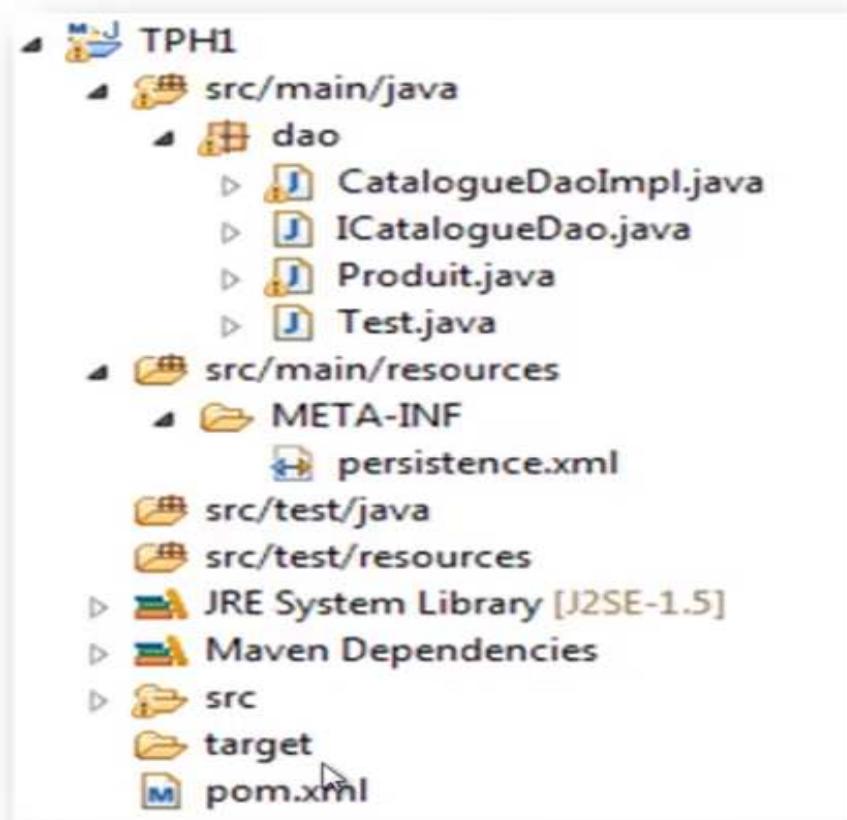
- **@Table**
 - Préciser le nom de la table concernée par le mapping. Par défaut c'est le nom de la classe qui sera considérée
- **@Column**
 - Associer un champ de la colonne à la propriété. Par défaut c'est le nom de la propriété qui sera considérée.
- **@Id**
 - Associer un champ de la table à la propriété en tant que clé primaire
- **@GeneratedValue**
 - Demander la génération automatique de la clé primaire au besoin
- **@Basic**
 - Représenter la forme de mapping la plus simple. Cette annotation est utilisée par défaut
- **@Transient**
 - Demander de ne pas tenir compte du champ lors du mapping
- **@OneToMany, @ManyToOne**
 - Pour décrire une association de type un à plusieurs et plusieurs à un
- **@JoinColumn**
 - Pour décrire une clé étrangère dans une table
- **@ManyToMany**
 - Pour décrire une association plusieurs à plusieurs
- **Etc...**

Application

- On souhaite créer une application qui permet de :
 - Ajouter un produit
 - Consulter tous les produits
 - Consulter les produits dont le nom contient un mot clé
 - Consulter un produit
 - Mettre à jour un produit
 - Supprimer un produit
- Un produit est défini par :
 - Sa référence de type Long (Auto incrémenté)
 - Sa désignation de type String
 - Son prix de type double
 - Sa quantité en stock
- On suppose que les produits sont stockés dans une base de données MySQL

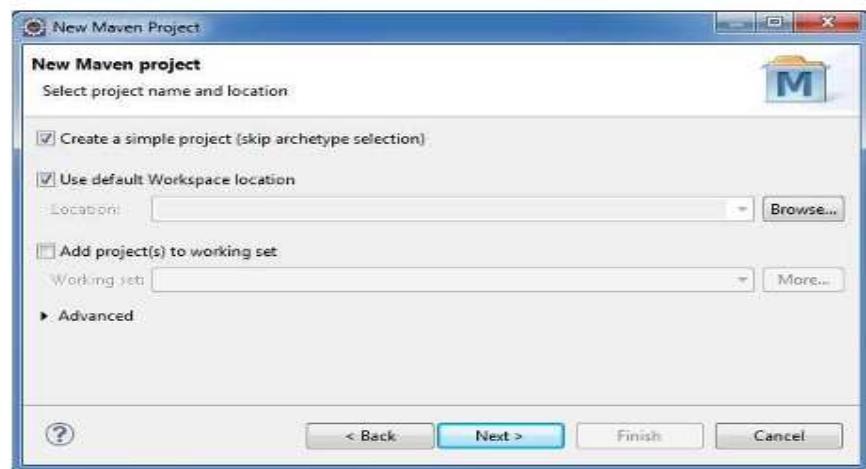
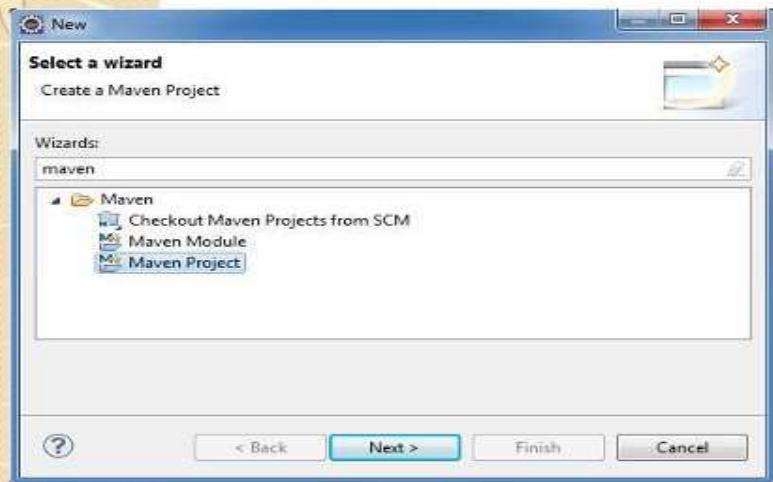


Structure du projet



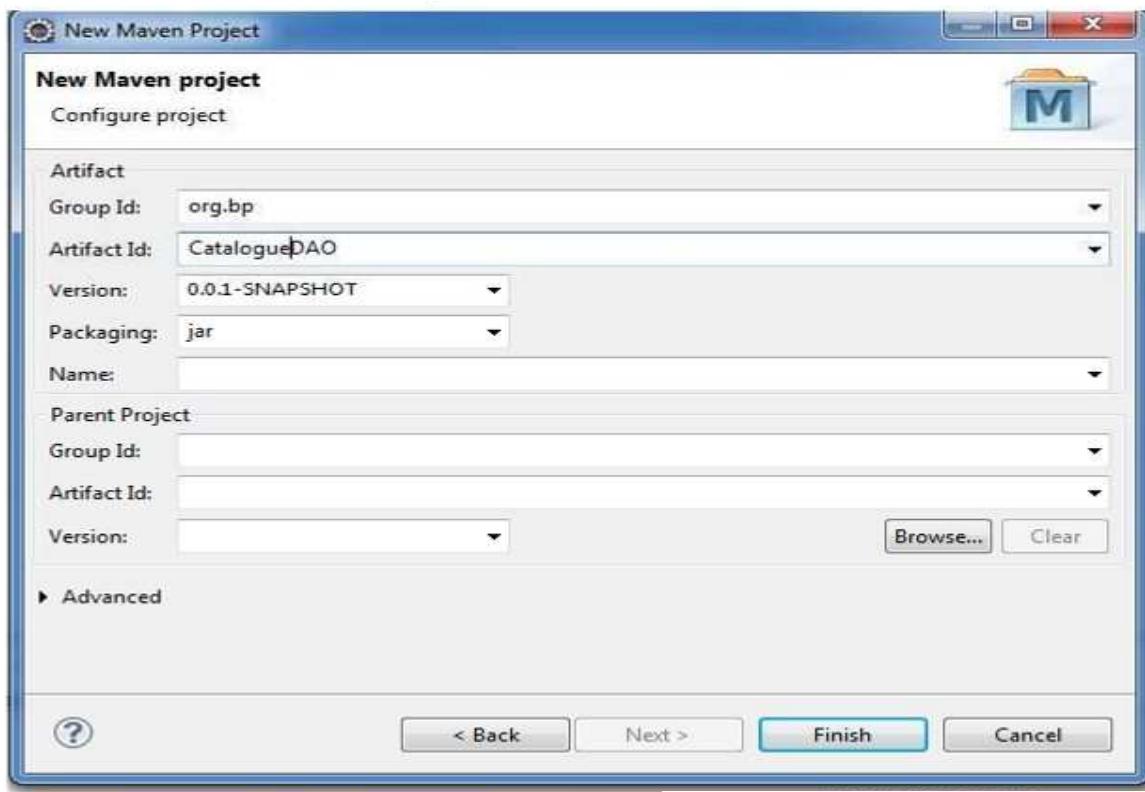
Projet Maven

- File>New>Maven>Maven Project



Paramètre du projet

- Group Id : org.bp
- Artifact Id : CatalogueDAO



Dépendances maven



Local Maven Repository

Utilisateurs > youssfi > .m2 > repository >	
Nom	Modifié le
classworlds	09/10/20
com	21/10/20
commons-cli	09/10/20
commons-io	09/10/20
commons-lang	14/10/20
commons-logging	14/10/20
dom4j	09/10/20
javax	21/10/20
junit	09/10/20
log4j	09/10/20
mysql	09/10/20
net	09/10/20
org	09/10/20

```
<dependencies>
    <!-- Hibernate --&gt;
    &lt;dependency&gt;
        &lt;groupId&gt;org.hibernate&lt;/groupId&gt;
        &lt;artifactId&gt;hibernate-core&lt;/artifactId&gt;
        &lt;version&gt;4.3.5.Final&lt;/version&gt;
    &lt;/dependency&gt;
    &lt;dependency&gt;
        &lt;groupId&gt;org.hibernate&lt;/groupId&gt;
        &lt;artifactId&gt;hibernate-entitymanager&lt;/artifactId&gt;
        &lt;version&gt;4.3.5.Final&lt;/version&gt;
    &lt;/dependency&gt;
    &lt;dependency&gt;
        &lt;groupId&gt;org.hibernate.javax.persistence&lt;/groupId&gt;
        &lt;artifactId&gt;hibernate-jpa-2.0-api&lt;/artifactId&gt;
        &lt;version&gt;1.0.1.Final&lt;/version&gt;
    &lt;/dependency&gt;
    <!-- MySQL Driver --&gt;
    &lt;dependency&gt;
        &lt;groupId&gt;mysql&lt;/groupId&gt;
        &lt;artifactId&gt;mysql-connector-java&lt;/artifactId&gt;
        &lt;version&gt;5.1.6&lt;/version&gt;
    &lt;/dependency&gt;
&lt;/dependencies&gt;</pre>
```

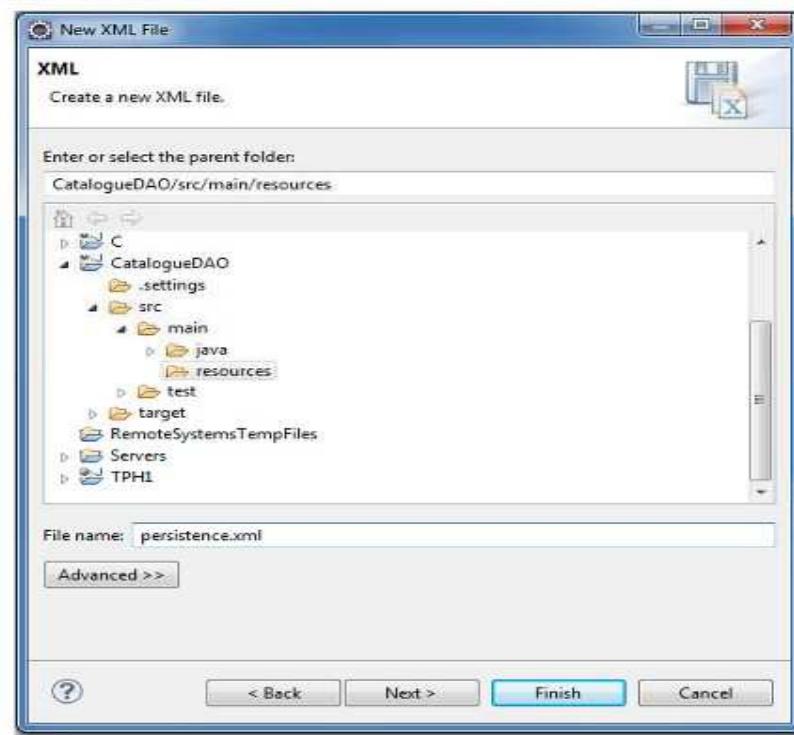
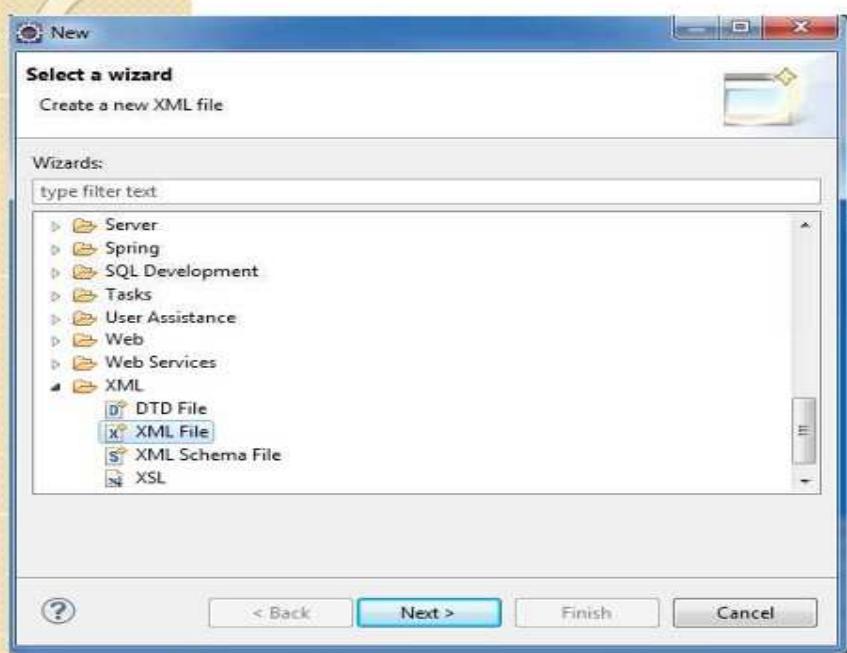
Entité Produit

```
package dao;  
import java.io.Serializable; import javax.persistence.*;  
@Entity  
@Table(name="PRODUITS")  
public class Produit implements Serializable {  
    @Id  
    @GeneratedValue(strategy=GenerationType.IDENTITY)  
    @Column(name="REF")  
    private Long reference;  
    @Column(name="DES")  
    private String designation;  
    private double prix;  
    private int quantite;  
    public Produit(String designation, double prix, int quantite) {  
        this.designation = designation; this.prix = prix; this.quantite = quantite;  
    }  
    public Produit() { }  
    // Getters et Setters  
}
```



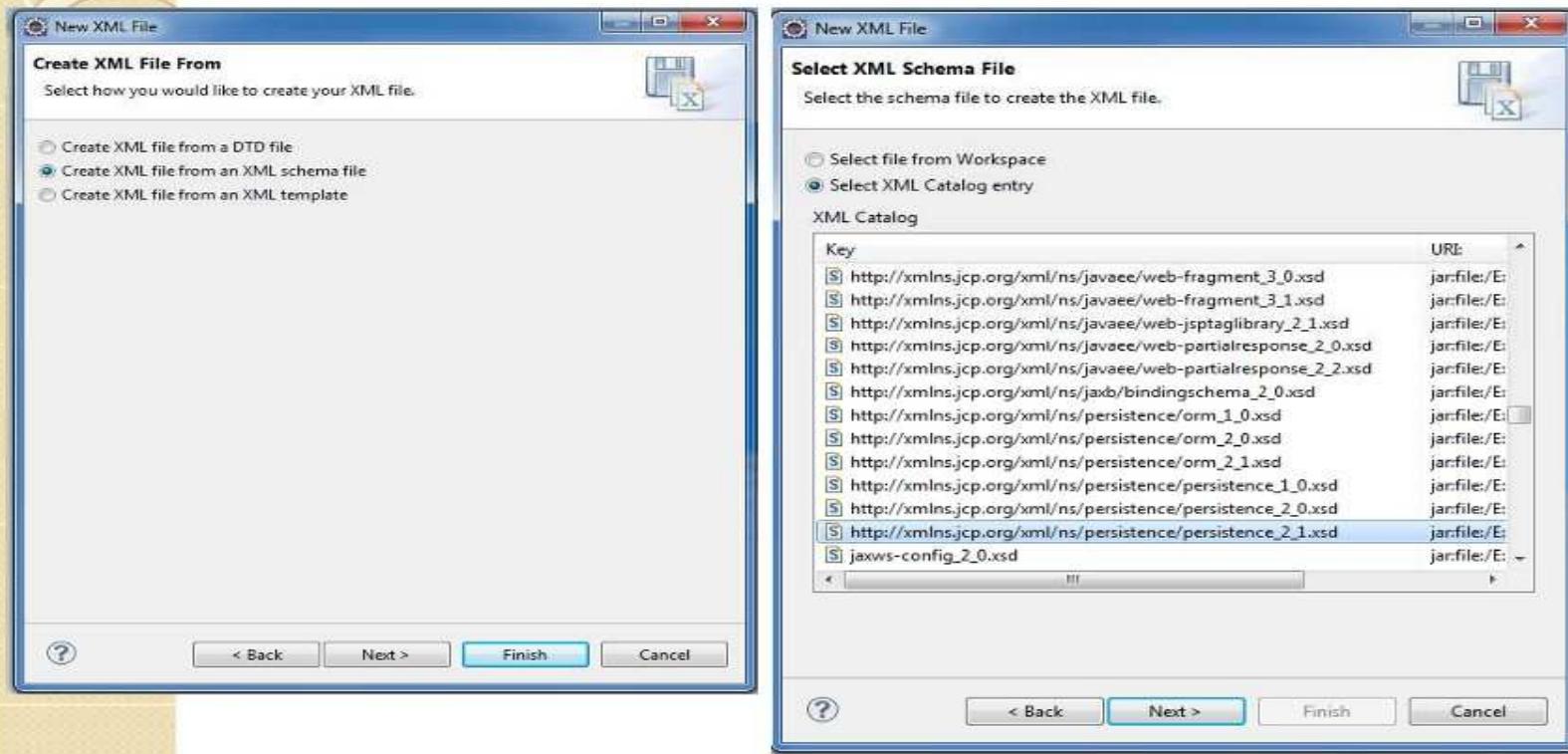
Unité de persistance : persistence.xml

- Crédation du fichier persistence.xml



Unité de persistance : persistence.xml

- Crédit de la formation



Configuration de l'unité de persistance : persistence.xml

```
<?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="UP_CAT" >
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <properties>
      <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/CAT"/>
      <property name="hibernate.connection.username" value="root"/>
      <property name="hibernate.connection.password" value="" />
      <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
    </properties>
  </persistence-unit>
</persistence>
```



Tester les entités

```
package dao;  
import javax.persistence.EntityManagerFactory;  
import javax.persistence.Persistence;  
public class TestEntities {  
    public static void main(String[] args) {  
        EntityManagerFactory entityManagerFactory=Persistence.createEntityManagerFactory("UP_CAT");  
    }  
}
```

- L'exécution de cette application permet de :
 - Créer un objet de type EntityManagerFactory.
 - Ce dernier lit le fichier de persistance « persistence.xml ».
 - Ce qui va entraîner la création du data source qui établit une connexion avec la base de données
 - Si les tables ne sont pas créées, entityManagerFactory génère les tables relatives aux entités. C'est ce qui est indiqué par la propriété :
 - <property name="hibernate.hbm2ddl.auto" value="update"/>

Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
REF	bigint(20)			Non	Aucune	AUTO_INCREMENT
DES	varchar(255)	latin1_swedish_ci		Oui	NULL	
prix	double			Non	Aucune	
quantite	int(11)			Non	Aucune	

Inreface ICatalogueDAO

```
package dao;

import java.util.List;

public interface ICatalogueDao {
    public void addProduit(Produit p);
    public List<Produit> listProduits();
    public List<Produit> produitsParMC(String mc);
    public Produit getProduit(Long idProduit);
    public void updateProduit(Produit p);
    public void deleteProduit(Long idP);
}
```



Gestion des entités par EntityManager

- EntityManager est une interface définie dans JPA.
- Chaque framework ORM possède sa propre implémentation de cette interface.
- EntityManager définit les méthodes qui permettent de gérer le cycle de vie de la persistance des Entity.
 - La méthode persist() permet rendre une nouvelle instance d'un EJB Entity persistante. Ce qui permet de sauvegarder son état dans la base de données
 - La méthode find() permet de charger une entité sachant sa clé primaire.
 - La méthode createQuery() permet de créer une requête EJBQL qui permet de charger une liste d'entités selon des critères.
 - La méthode remove() permet de programmer une entité persistante pour la suppression.
 - La méthode merge() permet de rendre une entité détachée persistante.

Gestion des entités : CatalogueDaoImpl

```
package dao;
import java.util.List; import javax.persistence.*;
public class CatalogueDaoImpl implements ICatalogueDao {
    /* Déclaration de l'objet EntityManager qui permet de gérer les entités*/
    private EntityManager entityManager;
    /* Constructeur */
    public CatalogueDaoImpl() {
        /* Création de l'objet Entity Manager Factory */
        EntityManagerFactory entityManagerFactory=
        Persistence.createEntityManagerFactory("UP_CAT");
        /* Création de l'objet Entity Manager */
        entityManager=entityManagerFactory.createEntityManager();
    }
}
```



Ajouter un produit : La méthode persist()

```
public void addProduit(Produit p) {  
    /* Création d'une transaction */  
    EntityTransaction transaction=entityManager.getTransaction();  
    /* Démarrer la transaction */  
    transaction.begin();  
    try {  
  
        /* enregister le produit p dans la base de données */  
        entityManager.persist(p);  
        /* Valider la transaction si tout se passe bien */  
  
        transaction.commit();  
    } catch (Exception e) {  
        /* Annuler la transaction en cas d'exception */  
        transaction.rollback();  
        e.printStackTrace();  
    }  
}
```



Consulter tous les produits : `createQuery()`

```
public List<Produit> listProduits() {  
    Query query=entityManager.createQuery("select p from Produit p");  
    return query.getResultList();  
}
```

- Pour sélectionner des données à partir de la base de données, on peut créer un objet `Query` en utilisant la méthode `createQuery()` de `entityManager`.
- La requête est spécifiée en utilisant le langage de requêtes JPA appelé **HQL** ou **JPA QL**.
- HQL ressemble à SQL, sauf que au lieu de des tables et des relations, entre les tables, on utilise les classes et les relations entre les classes.
- En fait, avec JPA, quant on fait la programmation orientée objet, on n'est pas sensé connaître la structure de la base de données, mais plutôt on connaît le diagramme de classes des différentes entités.
- C'est Hibernate qui va traduire le **HQL** en **SQL**. Ceci peut garantir à notre application de fonctionner correctement quelque soit le type de SGBD utilisé



Consulter tous les produits par mot clé : `createQuery()`

```
public List<Produit> produitsParMC(String mc) {  
    Query query=entityManager.createQuery("select p from Produit p  
    where p.designation like :x");  
    query.setParameter("x", "%" + mc + "%");  
    return query.getResultList();  
}
```



Consulter un produit : Méthode `find()`

```
public Produit getProduit(Long idProduit) {  
    Produit p=entityManager.find(Produit.class, idProduit);  
    return p;  
}
```

- Pour sélectionner un objet sachant son identifiant (clé primaire), on utilise la méthode `find()` de l'objet `entityManager`.
- Si l'objet n'existe pas, cette méthode retourne null.

Mettre à jour un produit : Méthode `merge()`

```
public void updateProduit(Produit p) {  
    entityManager.merge(p);  
}
```

- Pour mettre à jour un objet édité et modifier, on peut utiliser la méthode `merge()` de `entityManager`

Supprimer un produit : Méthode `remove()`

```
public void deleteProduit(Long idP) {  
    Produit p=entityManager.find(Produit.class, idP);  
    entityManager.remove(p);  
}
```

- Pour supprimer un objet sachant son identifiant, on peut utiliser conjointement les deux méthodes `find()` et `remove()` de `entityManager`

Tester les méthodes

```
package dao;
import java.util.List;
public class Test {
public static void main(String[] args) {
CatalogueDaoImpl dao=new CatalogueDaoImpl();
dao.addProduit(new Produit("P1", 8000, 4));
dao.addProduit(new Produit("P2", 6700, 2));
dao.addProduit(new Produit("P3", 5300, 1));
System.out.println("-----");
List<Produit> prods=dao.listProduits();
for(Produit p:prods){
System.out.println(p.getDesignation());
}}}
```

REF	DES	prix	quantite
1	P1	8000	4
2	P2	6700	2
3	P3	5300	1

Hibernate: insert into PRODUITS (DES, prix, quantite) values (?, ?, ?)

Hibernate: insert into PRODUITS (DES, prix, quantite) values (?, ?, ?)

Hibernate: insert into PRODUITS (DES, prix, quantite) values (?, ?, ?)

Hibernate: select produit0_.REF as REF1_0_, produit0_.DES as DES2_0_, produit0_.prix as prix3_0_,
produit0_.quantite as quantite4_0_ from PRODUITS produit0_

P1

P2

P3

Tester les méthodes

```
package dao;
import java.util.List;
public class Test {
public static void main(String[] args) {
CatalogueDaoImpl dao=new CatalogueDaoImpl();
System.out.println("-----");
System.out.println("Consulter Les produits par mot clé");
List<Produit> prods2=dao.produitsParMC("P");
for(Produit p:prods2){
System.out.println(p.getDesignation());
}}}
```

REF	DES	prix	quantite
1	P1	8000	4
2	P2	6700	2
3	P3	5300	1

Consulter les produits par mot clé

Hibernate: select produit0_.REF as REFI_0_,produit0_.DES as DES2_0_,produit0_.prix as prix3_0_,produit0_.quantite as quantite4_0_ from PRODUITS produit0_ where produit0_.DES like ?

P1

P2

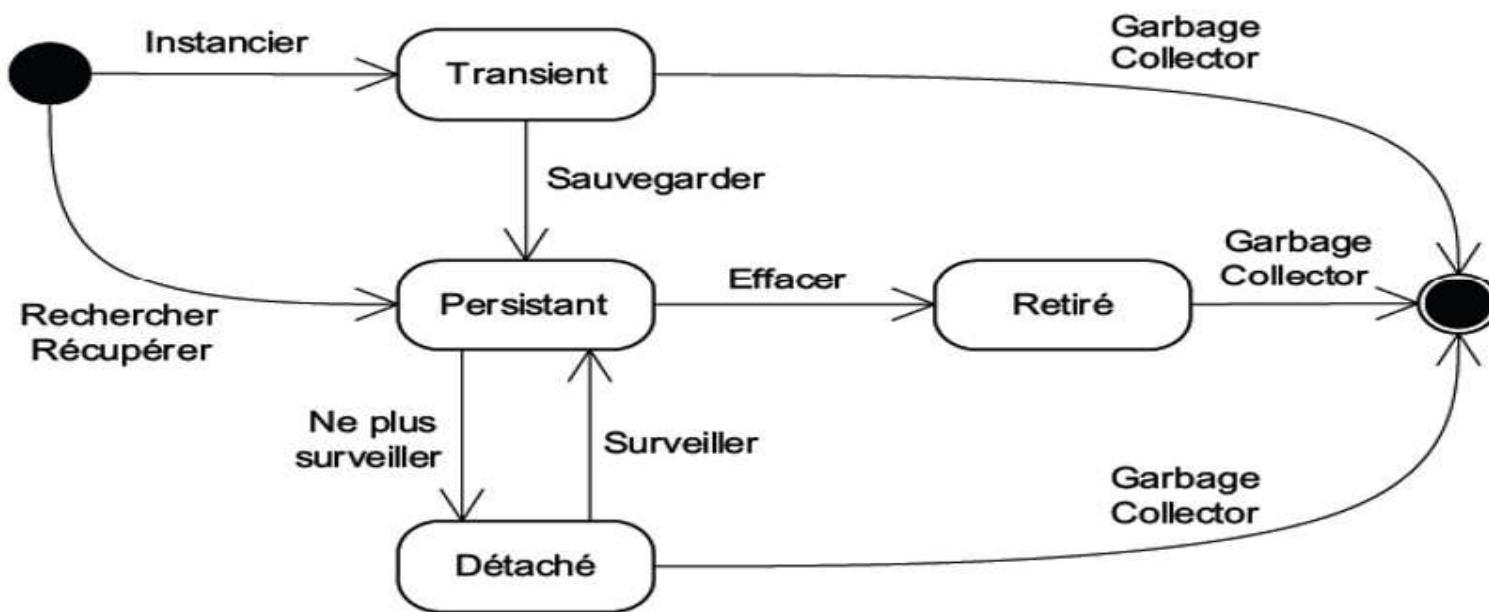
P3

Tester les méthodes

```
package dao;  
import java.util.List;  
public class Test {  
public static void main(String[] args) {  
CatalogueDaoImpl dao=new CatalogueDaoImpl();  
System.out.println("-----");  
System.out.println("Consulter un produit");  
Produit p=dao.getProduit(1L);  
System.out.println(p.getDesignation());  
System.out.println(p.getPrix());  
System.out.println("-----");  
System.out.println("Modifier le prix du produit");  
p.setPrix(1234);  
dao.updateProduit(p);  
System.out.println("-----");  
System.out.println("Supprimer un produit");  
dao.deleteProduit(3L);  
}  
}
```

REF	DES	prix	quantite
1	P1	1234	4
2	P2	6700	2

Cycle de vie d'un EJB Entity



Objet Persistant

- Un objet *persistent* est un objet qui possède son image dans le datastore et dont la durée de vie est potentiellement infinie.
- Pour garantir que les modifications apportées à un objet sont rendues persistantes, c'est-à-dire sauvegardées, l'objet est surveillé par un « traqueur » d'instances persistantes.
- Ce rôle est joué par le gestionnaire d'entités.

Etat Transient

- Un objet *transient* est un objet qui n'a pas son image stockée dans le datastore.
- Il s'agit d'un objet « temporaire », qui meurt lorsqu'il n'est plus utilisé par personne. En Java, le garbage collector le ramasse lorsque aucun autre objet ne le référence.

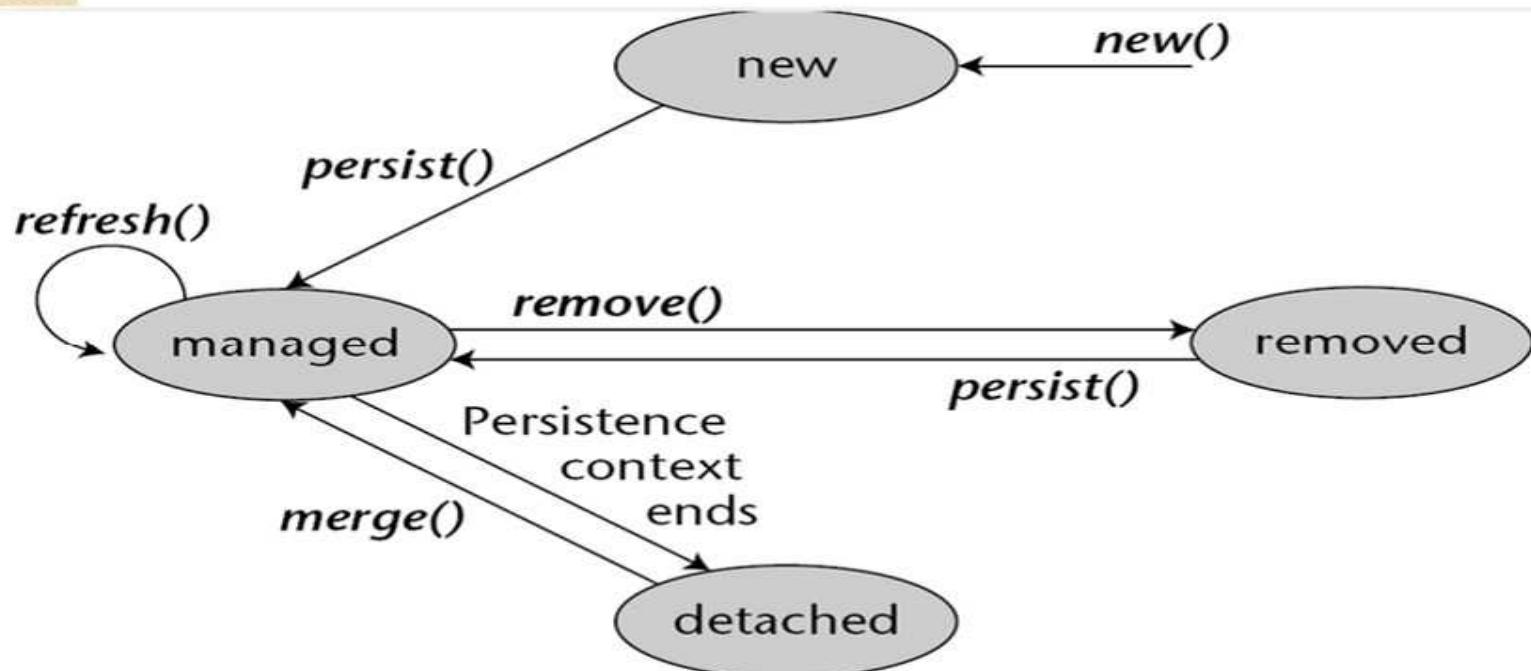
Etat Détaché

- Un objet détaché est un objet qui possède son image dans le datastore mais qui échappe temporairement à la surveillance opérée par le gestionnaire d'entités.
- Pour que les modifications potentiellement apportées pendant cette phase de détachement soient enregistrées, il faut effectuer une opération manuelle pour *merger* cette instance au gestionnaire d'entités.

Etat Retiré

- Un objet *retiré* est un objet actuellement géré par le gestionnaire d'entités mais programmé pour ne plus être persistant.
- À la validation de l'unité de travail, un ordre SQL delete sera exécuté pour retirer son image du datastore.

Cycle de vie d'un EJB Entity



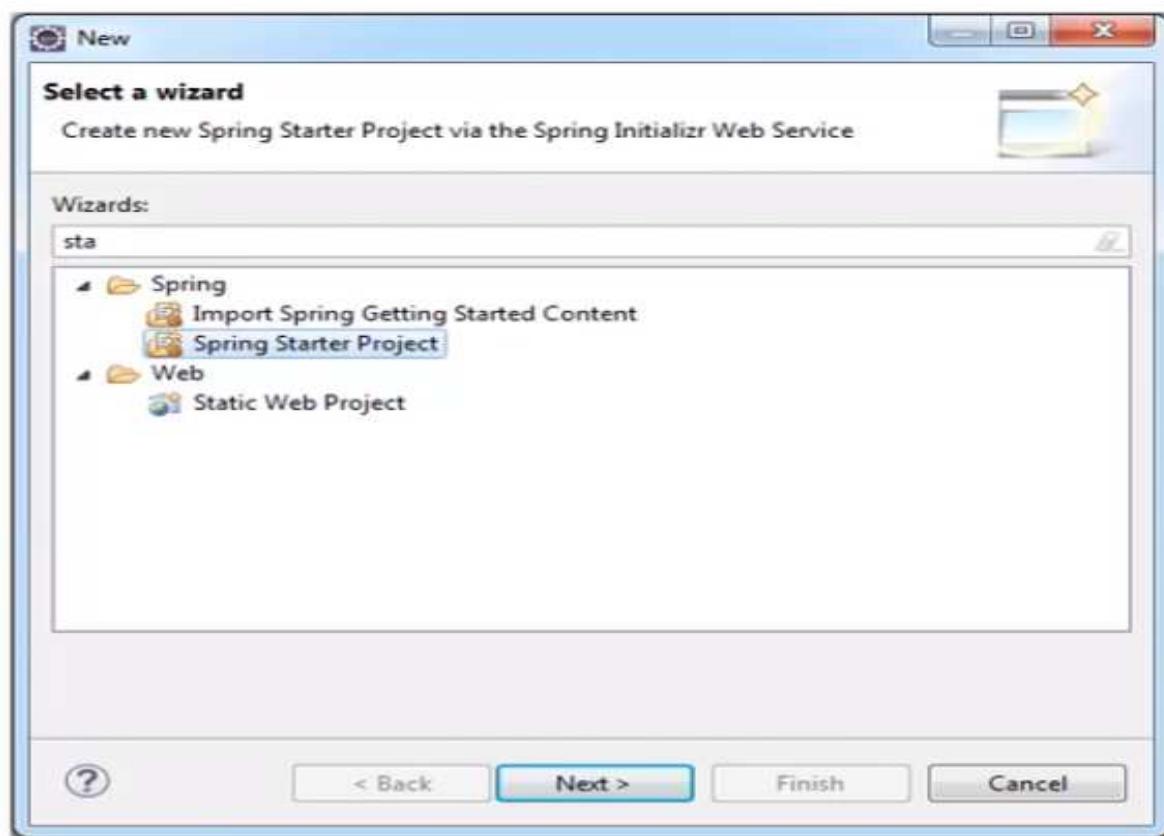


- **JPA DANS UN PROJET
SPRING BOOT**

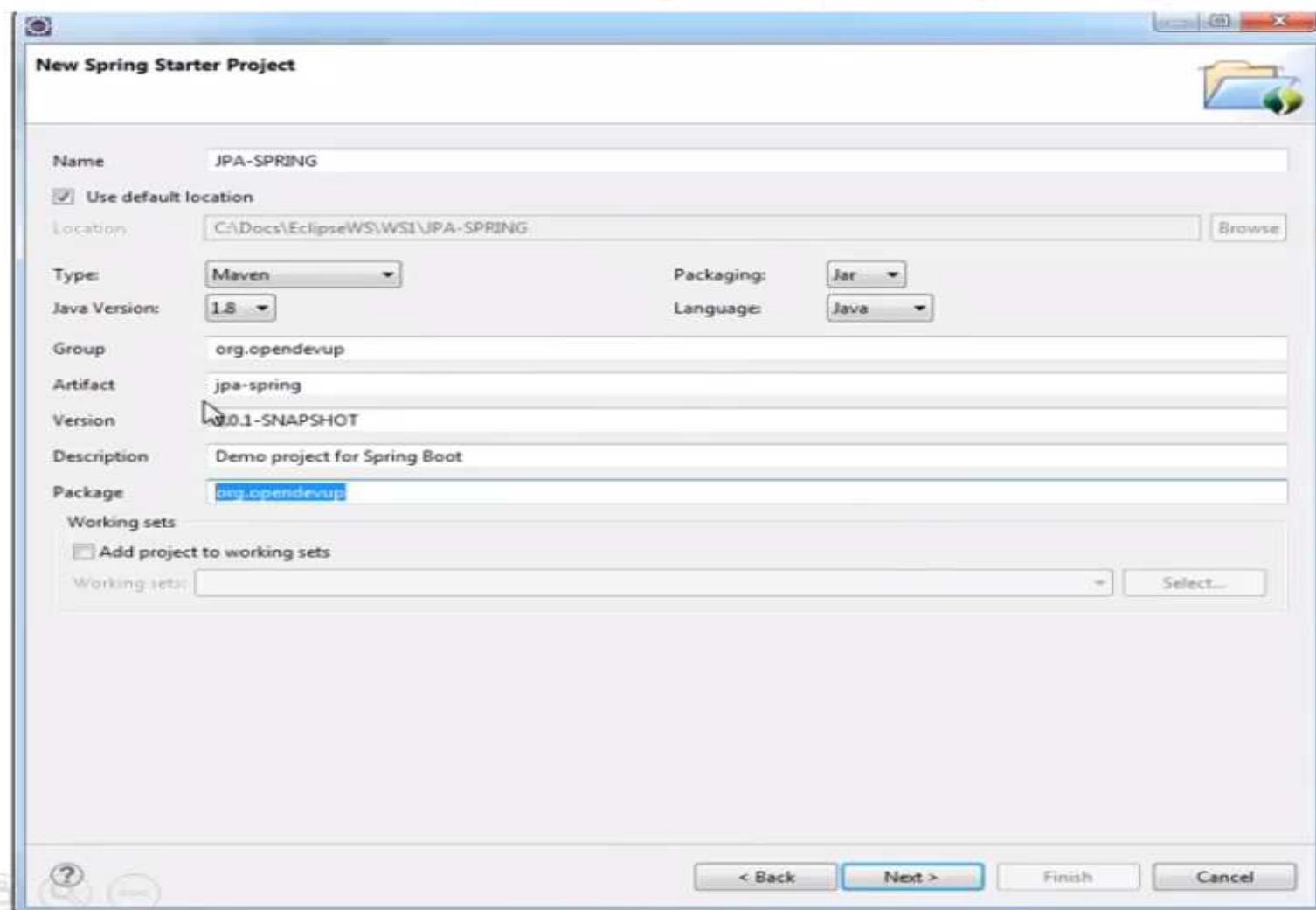
JPA dans un projet Spring

- Spring est Framework qui assure l'inversion de contrôle.
- Spring peut s'occuper du code technique comme la configuration de JPA et la gestion des transactions.
- Avec Spring on peut simplifier le code de notre application en lui déléguant des tâches techniques.
- Spring Boot est une version de spring qui permet de simplifier à l'extrême, entre autres:
 - La gestion des dépendances maven
 - l'injection des dépendances (Principe de zero config)

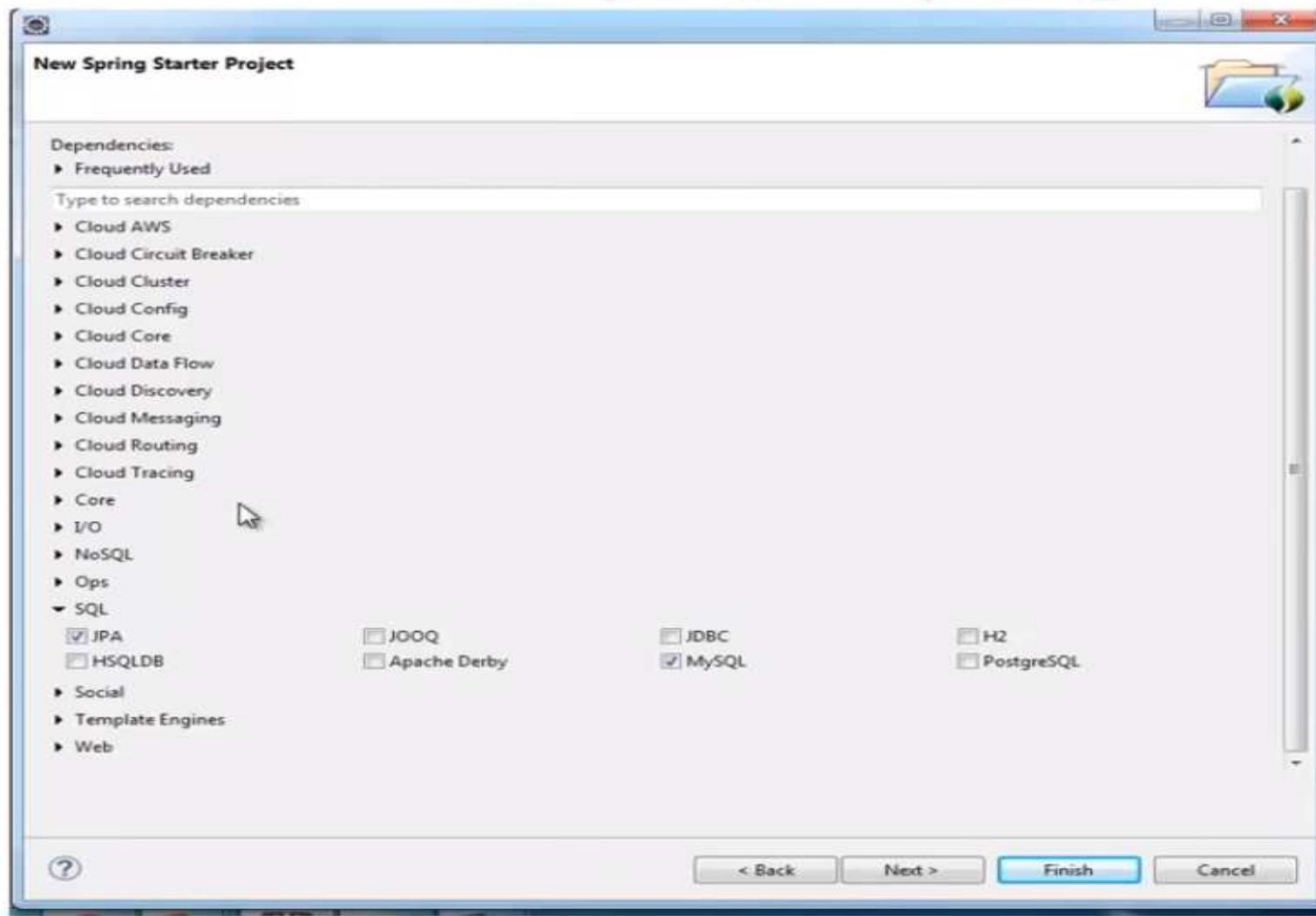
Création d'un projet Spring Boot



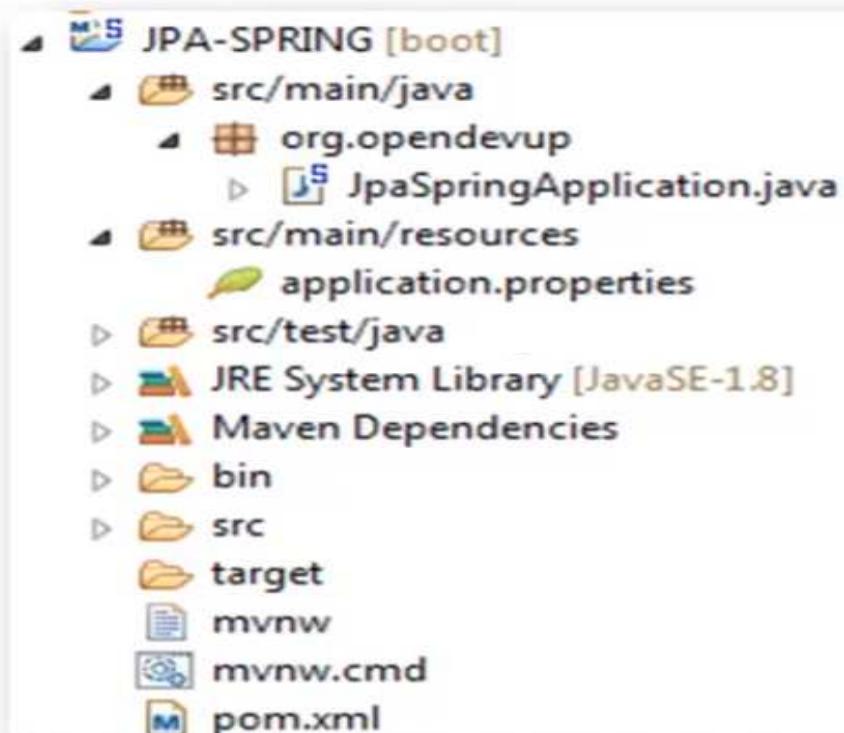
Création d'un projet Spring Boot



Création d'un projet Spring Boot



Structure du projet



Dépendances Maven

Maven Dependencies

- ▷ spring-boot-starter-data-jpa-1.3.3.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\boot\starter\spring-boot-starter-data-jpa\1.3.3.RELEASE\spring-boot-starter-data-jpa-1.3.3.RELEASE.jar
- ▷ spring-boot-starter-1.3.3.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\boot\starter\spring-boot-starter\1.3.3.RELEASE\spring-boot-starter-1.3.3.RELEASE.jar
- ▷ spring-boot-1.3.3.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\boot\spring-boot\1.3.3.RELEASE\spring-boot-1.3.3.RELEASE.jar
- ▷ spring-boot-autoconfigure-1.3.3.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\boot\autoconfigure\1.3.3.RELEASE\spring-boot-autoconfigure-1.3.3.RELEASE.jar
- ▷ spring-boot-starter-logging-1.3.3.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\boot\starter\logging\1.3.3.RELEASE\spring-boot-starter-logging-1.3.3.RELEASE.jar
- ▷ logback-classic-1.1.5.jar - C:\Users\youssf\l.m2\repository\ch\qos\logback\logback-classic\1.1.5\logback-classic-1.1.5.jar
- ▷ logback-core-1.1.5.jar - C:\Users\youssf\l.m2\repository\ch\qos\logback\logback-core\1.1.5\logback-core-1.1.5.jar
- ▷ jul-to-slf4j-1.7.16.jar - C:\Users\youssf\l.m2\repository\org\slf4j\jul-to-slf4j\1.7.16\jul-to-slf4j-1.7.16.jar
- ▷ log4j-over-slf4j-1.7.16.jar - C:\Users\youssf\l.m2\repository\org\log4j\log4j-over-slf4j\1.7.16\log4j-over-slf4j-1.7.16.jar
- ▷ snakeyaml-1.16.jar - C:\Users\youssf\l.m2\repository\org\yaml\snakeyaml\1.16\snakeyaml-1.16.jar
- ▷ spring-boot-starter-aop-1.3.3.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\boot\starter\spring-boot-starter-aop\1.3.3.RELEASE\spring-boot-starter-aop-1.3.3.RELEASE.jar
- ▷ spring-aop-4.2.5.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\boot\dependency\spring-aop\4.2.5.RELEASE\spring-aop-4.2.5.RELEASE.jar
- ▷ aopalliance-1.0.jar - C:\Users\youssf\l.m2\repository\javaee\api\aopalliance\1.0\javaee-api-1.0.jar
- ▷ aspectjweaver-1.8.8.jar - C:\Users\youssf\l.m2\repository\org\aspectj\aspectjweaver\1.8.8\aspectjweaver-1.8.8.jar
- ▷ spring-boot-starter-jdbc-1.3.3.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\boot\starter\spring-boot-starter-jdbc\1.3.3.RELEASE\spring-boot-starter-jdbc-1.3.3.RELEASE.jar
- ▷ tomcat-jdbc-8.0.32.jar - C:\Users\youssf\l.m2\repository\org\apache\tomcat\tomcat-jdbc\8.0.32\tomcat-jdbc-8.0.32.jar
- ▷ tomcat-juli-8.0.32.jar - C:\Users\youssf\l.m2\repository\org\apache\tomcat\tomcat-juli\8.0.32\tomcat-juli-8.0.32.jar
- ▷ spring-jdbc-4.2.5.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-jdbc\4.2.5.RELEASE\spring-jdbc-4.2.5.RELEASE.jar
- ▷ hibernate-entitymanager-4.3.11.Final.jar - C:\Users\youssf\l.m2\repository\org\hibernate\entitymanager\4.3.11.Final\hibernate-entitymanager-4.3.11.Final.jar
- ▷ jboss-logging-3.3.0.Final.jar - C:\Users\youssf\l.m2\repository\org\jboss\jboss-logging\3.3.0.Final\jboss-logging-3.3.0.Final.jar
- ▷ jboss-logging-annotations-1.2.0.Beta1.jar - C:\Users\youssf\l.m2\repository\org\jboss\jboss-logging-annotations\1.2.0.Beta1\jboss-logging-annotations-1.2.0.Beta1.jar
- ▷ hibernate-core-4.3.11.Final.jar - C:\Users\youssf\l.m2\repository\org\hibernate\hibernate-core\4.3.11.Final\hibernate-core-4.3.11.Final.jar
- ▷ antlr-2.7.7.jar - C:\Users\youssf\l.m2\repository\antlr\antlr\2.7.7\antlr-2.7.7.jar
- ▷ jandex-1.1.0.Final.jar - C:\Users\youssf\l.m2\repository\org\jboss\jandex\jandex-1.1.0.Final\jandex-1.1.0.Final.jar
- ▷ dom4j-1.6.1.jar - C:\Users\youssf\l.m2\repository\dom4j\dom4j\1.6.1\dom4j-1.6.1.jar

- ▷ xml-apis-1.0.b2.jar - C:\Users\youssf\l.m2\repository\xml-apis\xml-apis-1.0.b2\xml-apis-1.0.b2.jar
- ▷ hibernate-commons-annotations-4.0.5.Final.jar - C:\Users\youssf\l.m2\repository\org\hibernate\hibernate-commons-annotations\4.0.5.Final\hibernate-commons-annotations-4.0.5.Final.jar
- ▷ hibernate-jpa-2.1-api-1.0.0.Final.jar - C:\Users\youssf\l.m2\repository\org\hibernate\hibernate-jpa-2.1-api\1.0.0.Final\hibernate-jpa-2.1-api-1.0.0.Final.jar
- ▷ javassist-3.18.1-GA.jar - C:\Users\youssf\l.m2\repository\javassist\javassist\3.18.1-GA\javassist-3.18.1-GA.jar
- ▷ javax.transaction-api-1.2.jar - C:\Users\youssf\l.m2\repository\javax\transaction\javax.transaction-api\1.2\javax.transaction-api-1.2.jar
- ▷ spring-data-jpa-1.9.4.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-data\jpa\1.9.4.RELEASE\spring-data-jpa-1.9.4.RELEASE.jar
- ▷ spring-data-commons-1.11.4.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-data\commons\1.11.4.RELEASE\spring-data-commons-1.11.4.RELEASE.jar
- ▷ spring-orm-4.2.5.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-orm\4.2.5.RELEASE\spring-orm-4.2.5.RELEASE.jar
- ▷ spring-context-4.2.5.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-context\4.2.5.RELEASE\spring-context-4.2.5.RELEASE.jar
- ▷ spring-expression-4.2.5.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-expression\4.2.5.RELEASE\spring-expression-4.2.5.RELEASE.jar
- ▷ spring-tx-4.2.5.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-tx\4.2.5.RELEASE\spring-tx-4.2.5.RELEASE.jar
- ▷ spring-beans-4.2.5.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-beans\4.2.5.RELEASE\spring-beans-4.2.5.RELEASE.jar
- ▷ slf4j-api-1.7.16.jar - C:\Users\youssf\l.m2\repository\org\slf4j\slf4j-api\1.7.16\slf4j-api-1.7.16.jar
- ▷ jcl-over-slf4j-1.7.16.jar - C:\Users\youssf\l.m2\repository\org\log4j\jcl-over-slf4j\1.7.16\jcl-over-slf4j-1.7.16.jar
- ▷ spring-aspects-4.2.5.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-aspects\4.2.5.RELEASE\spring-aspects-4.2.5.RELEASE.jar
- ▷ mysql-connector-java-5.1.38.jar - C:\Users\youssf\l.m2\repository\mysql\mysql-connector-java\5.1.38\mysql-connector-java-5.1.38.jar
- ▷ spring-boot-starter-test-1.3.3.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\boot\dependency\spring-boot-starter-test\1.3.3.RELEASE\spring-boot-starter-test-1.3.3.RELEASE.jar
- ▷ junit-4.12.jar - C:\Users\youssf\l.m2\repository\junit\junit\4.12\junit-4.12.jar
- ▷ mockito-core-1.10.19.jar - C:\Users\youssf\l.m2\repository\org\mockito\mockito-core\1.10.19\mockito-core-1.10.19.jar
- ▷ objenesis-2.1.jar - C:\Users\youssf\l.m2\repository\org\objenesis\objenesis\2.1\objenesis-2.1.jar
- ▷ hamcrest-core-1.3.jar - C:\Users\youssf\l.m2\repository\org\hamcrest\hamcrest-core\1.3\hamcrest-core-1.3.jar
- ▷ hamcrest-library-1.3.jar - C:\Users\youssf\l.m2\repository\org\hamcrest\hamcrest-library\1.3\hamcrest-library-1.3.jar
- ▷ spring-core-4.2.5.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-core\4.2.5.RELEASE\spring-core-4.2.5.RELEASE.jar
- ▷ spring-test-4.2.5.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-test\4.2.5.RELEASE\spring-test-4.2.5.RELEASE.jar



pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>org.opendevup</groupId>
<artifactId>jpa-spring</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>
<name>JPA-SPRING</name>
<description>Demo project for Spring Boot</description>
<parent>
    <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.3.3.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <java.version>1.8</java.version>
</properties>
```

pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>
```



application.properties

```
spring.datasource.url = jdbc:mysql://localhost:3306/db_cat_mvc
spring.datasource.username = root
spring.datasource.password =
spring.datasource.driverClassName = com.mysql.jdbc.Driver
spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
```

Entité Produit

```
package org.opendevup.entities;

import java.io.Serializable;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
@Entity
public class Produit implements Serializable {
@Id @GeneratedValue
    private Long idProduit;
    private String designation;
    private double prix;

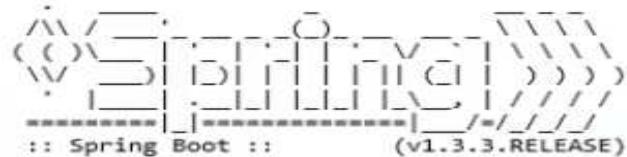
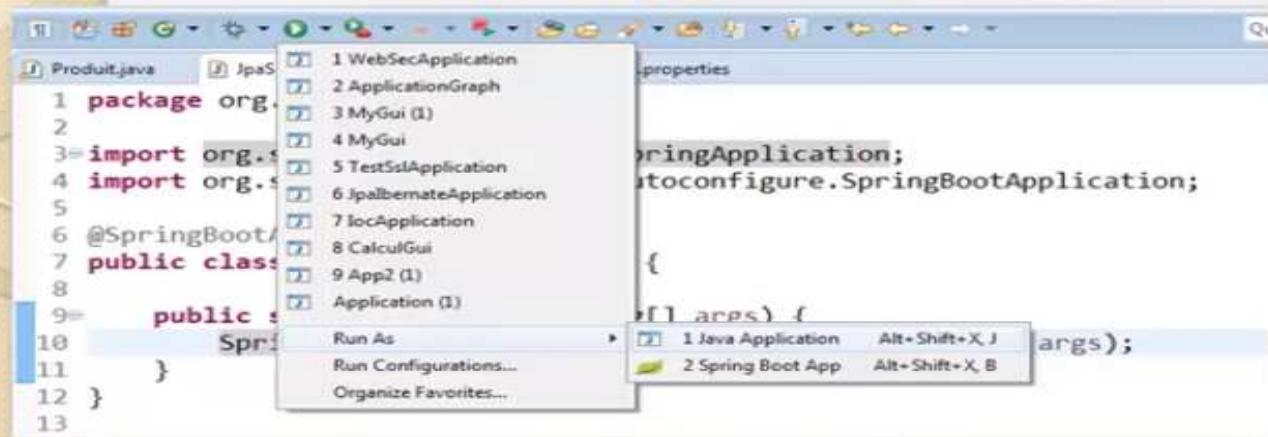
    // Constructeurs
    // Getters et Setters
}
```



Application Spring Boot

```
package org.opendevup;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class JpaSpringApplication {
    public static void main(String[] args) {
        SpringApplication.run(JpaSpringApplication.class, args);
    }
}
```

Démarrer l'application Spring



```
:: Spring Boot ::      (v1.3.3.RELEASE)

main] org.hibernate.tool.hbm2ddl.SchemaUpdate : HHH000228: Running hbm2ddl schema update
main] org.hibernate.tool.hbm2ddl.SchemaUpdate : HHH000102: Fetching database metadata
main] org.hibernate.tool.hbm2ddl.SchemaUpdate : HHH000396: Updating schema
main] java.sql.DatabaseMetaData          : HHH000262: Table not found: produit
main] java.sql.DatabaseMetaData          : HHH000262: Table not found: produit
main] java.sql.DatabaseMetaData          : HHH000262: Table not found: produit
main] org.hibernate.tool.hbm2ddl.SchemaUpdate : HHH000232: Schema update complete
main] o.s.j.e.a.AnnotationMBeanExporter   : Registering beans for JMX exposure on startup
main] org.opendevup.JpaSpringApplication : Started JpaSpringApplication in 4.88 seconds (JVM running for 5.935)
Thread-2] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
```

Structure de la table
Produits générée

	Colonne	Type	Interclassement	Attributs	Null	Défaut	Extra
	id_produit	bigint(20)			Non	Aucun	AUTO_INCREMENT
	designation	varchar(255)	latin1_swedish_ci		Oui	NULL	
	prix	double			Non	Aucun	

Interface IProduitDao

```
package org.opendevup.dao;
import java.util.List;
import org.opendevup.entities.Produit;
public interface IProduitDao {
    public Produit save(Produit p);
    public List<Produit> findAll();
    public Produit findOne(Long id);
    public void remove(Long id);
    public void update(Produit p);
    public List<Produit> findByDesignation(String des);
}
```

Implémentation ProduitDaoImpl

```
package org.opendevup.dao;

import java.util.List; import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext; import javax.persistence.Query;
import javax.transaction.Transactional; import org.opendevup.entities.Produit;
import org.springframework.stereotype.Repository;

@Repository
@Transactional
public class ProduitDaoImpl implements IProduitDao {

    @PersistenceContext
    private EntityManager entityManager;

    @Override
    public Produit save(Produit p) {
        entityManager.persist(p);
        return p;
    }
}
```



Implémentation ProduitDaolmpl

```
@Override
public List<Produit> findAll() {
    Query req=entityManager.createQuery("select p from Produit p");
    return req.getResultList();
}
@Override
public Produit findOne(Long id) {
    Produit p=entityManager.find(Produit.class, id);
    return p;
}
@Override
public void remove(Long id) {
    Produit p=entityManager.find(Produit.class, id);
    entityManager.remove(p);
}
```

Implémentation ProduitDaoImpl

```
@Override
public List<Produit> findByDesignation(String des) {
    Query req=entityManager.createQuery("select p from Produit p
where p.designation like :x");
    req.setParameter("x", des);
    return req.getResultList();
}

@Override
public void update(Produit p) {
    entityManager.merge(p);
}

public ProduitDaoImpl() {
    System.out.println("*****");
    System.out.println("Instaciation de CatalogueDaoImpl");
    System.out.println("*****");
}
}
```

Tester les méthodes

```
package org.opendevup;
import java.util.List; import org.opendevup.dao.ICatalogueDao;
import org.opendevup.entities.Produit; import
org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;

@SpringBootApplication
public class JpaSpringApplication {
    public static void main(String[] args) {
        ApplicationContext ctx=SpringApplication.run(JpaSpringApplication.class, args);
        System.out.println("-----");
        IProduitDao dao=ctx.getBean(IProduitDao.class);
        System.out.println("Ajouter des produits");
        dao.save(new Produit("AX 453", 870));
        dao.save(new Produit("TA 6753", 1230));
        dao.save(new Produit("HP 153", 340));
```

Tester les méthodes

```
System.out.println("Consulter tous les produits");
List<Produit> prods=dao.findAll();
for(Produit p:prods){
    System.out.println(p.getDesignation()+"----"+p.getPrix());
}

System.out.println("Consulter Un produit");
Produit p=dao.findOne(2L);
System.out.println("Produit :"+p.getDesignation());

System.out.println("Mettre à jour Le produit 2");
p.setDesignation("Epson A76500");
dao.update(p);

System.out.println("Afficher les produits dont la désignation contient A ");
List<Produit> listProduits=dao.findByDesignation("%A%");
for(Produit pr:listProduits){
    System.out.println(pr.getDesignation()+"----"+pr.getPrix());
}
```



- **SPRING DATA**

JPA, Hibernate, Spring Data

- Spring Data est un module de Spring qui a déjà créé des interfaces génériques et des implémentations génériques qui permettent de gérer les entités JPA.
- En utilisant Spring Data, vous n'aurez plus besoin de faire appel à l'objet EntityManager pour gérer la persitence. Spring Data le fait à votre place.
- Spring Data nous évite de créer les interfaces et les implémentation JPA de la couche DAO.
- Il suffit de créer une interface qui hérite de l'interface JpaRepository pour hériter toutes les méthodes classiques qui permettent de gérer les entités JPA.
- En cas de besoin, vous avez la possibilité d'ajouter d'autres méthodes en les déclarant à l'intérieur de l'interface JpaRepository, sans avoir besoin de les implémenter. Spring Data le fera à votre place.



Exemple d'interface JPARepository

```
package org.sid.dao; import java.util.List;  
import org.sid.entities.Produit;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.query.Param;  
public interface ProduitRepository  
    extends JpaRepository<Produit, Long> {  
}
```

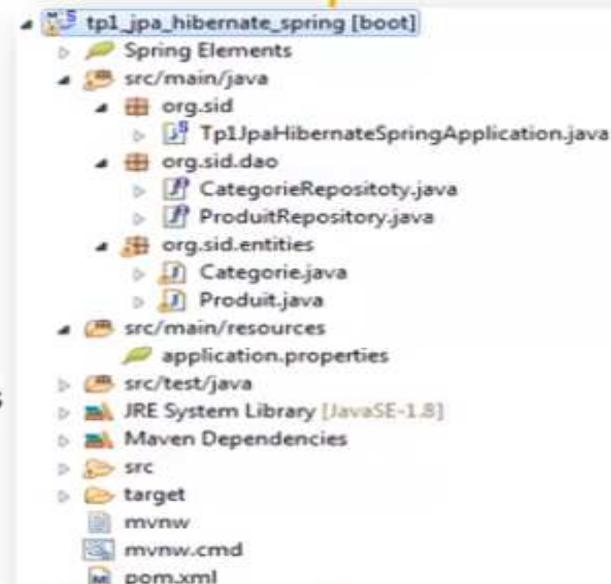
Exemple d'interface JPARespository

```
package org.sid.dao; import java.util.List;
import org.sid.entities.Produit;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
public interface ProduitRepository
    extends JpaRepository<Produit, Long> {
    @Query("select p from Produit p where p.designation like :x and p.prix>:y")
    public List<Produit> chercherProduits(
        @Param("x")String mc,
        @Param("y")double prixMin);
}
```



Exemple d'application Spring Boot

```
@SpringBootApplication
public class Tp1JpaHibernateSpringApplication {
    public static void main(String[] args) {
        ApplicationContext ctx=
        SpringApplication.run(Tp1JpaHibernateSpringApplication.class, args);
        CategorieRepositoty categorieRepositoty=
        ctx.getBean(CategorieRepositoty.class);
        categorieRepositoty.save(new Categorie("Ordinateurs"));
        categorieRepositoty.save(new Categorie("Imprimantes"));
        categorieRepositoty.save(new Categorie("Logiciels"));
        ProduitRepository produitRepository=ctx.getBean(ProduitRepository.class);
        Categorie c1=categorieRepositoty.findOne(1L);
        Categorie c2=categorieRepositoty.findOne(2L);
        produitRepository.save(new Produit("LX 564", 980, 34, c1));
        produitRepository.save(new Produit("HP 564", 45, 4, c1));
        produitRepository.save(new Produit("HP 76 564", 11, 12, c2));
        List<Produit> produits=produitRepository.findAll();
        for(Produit p:produits){ System.out.println(p.getDesignation()); }
        System.out.println("*****");
        List<Produit> prs=produitRepository.chercherProduits("%H%",11);
        for(Produit p:prs){ System.out.println(p.getDesignation()); }
    }
}
```





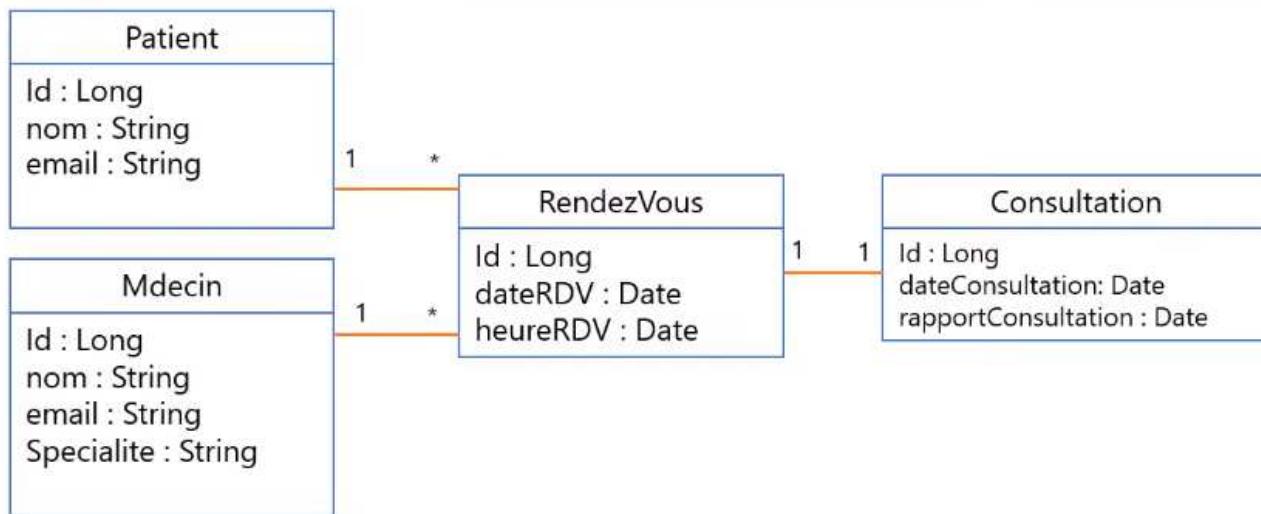
Gérer les associations et l'héritage entre les entités

- Associations
 - `@OneToMany`
 - `@ManyToOne`
 - `@ManyToMany`
 - `@OneToOne`
- Héritage
 - Une table par hiérarchie
 - Une table pour chaque classe concrète
 - Une table pour la classe parente et une table pour chaque classe fille



Association OneToMany, ManyToOne, OneToOne : Exemple Rendez-Vous Médecins , Patients

- ▶ On souhaite gérer les rendez-vous des consultations des patients effectuées par des médecins.
- ▶ Chaque Rendez-vous concerne un patient et un médecin.
- ▶ Pour chaque rendez-vous on associe une seule consultation issue de rendez-vous.
- ▶ Un Patient peut prendre plusieurs rendez-vous



Association OneToMany, ManyToOne, OneToOne : Exemple Rendez-Vous Médecins , Patients

```
@Entity
public class Patient {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private String email;
    @OneToMany(mappedBy = "patient")
    private Collection<RendezVous> rendezVous;
}
```

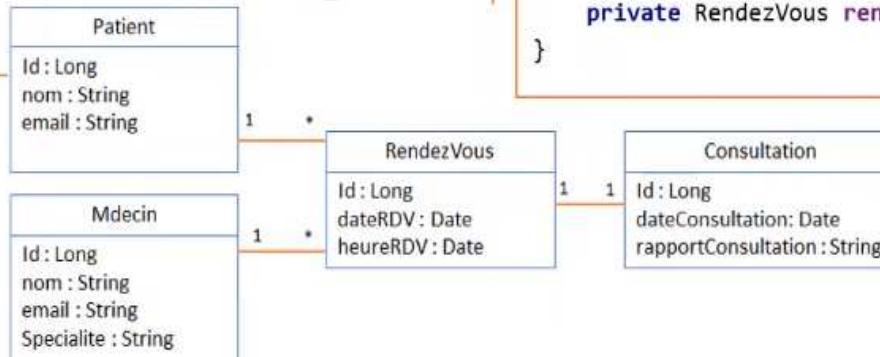
PATIENT
+ ID
+ EMAIL
+ NOM

```
@Entity
public class Medecin {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private String specialite;
    private String email;
    @OneToMany(mappedBy = "medecin")
    private Collection<RendezVous> rendezVous;
}
```

MEDECIN
+ ID
+ EMAIL
+ NOM
+ SPECIALITE

```
@Entity
public class RendezVous {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date dateRendezVous;
    @ManyToOne
    private Medecin medecin;
    @ManyToOne
    private Patient patient;
    @OneToOne
    private Consultation consultation;
}
```

RENDEZ_VOUS
+ ID
+ DATE_RENDEZ_VOUS
+ CONSULTATION_ID
+ MEDECIN_ID
+ PATIENT_ID



```
@Entity
public class Consultation {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date dateConsultation;
    private String rapportConsultation;
    private double prixConsultation;
    @OneToOne(mappedBy = "consultation")
    private RendezVous rendezVous;
}
```

CONSULTATION
+ ID
+ DATE_CONSULTATION
+ PRIX_CONSULTATION
+ RAPPORT_CONSULTATION

Cas de ManyMany

- On suppose que l'on souhaite de créer une application qui permet de gérer des Utilisateurs appartenant à des groupes. Chaque Groupe peut contenir plusieurs utilisateurs.



```
@Entity
public class Utilisateur {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(unique = true)
    private String userName;
    private String password;
    @ManyToMany(mappedBy = "utilisateurs", fetch = FetchType.EAGER)
    private Collection<Groupe> groupes=new ArrayList<>();
}
```

```
@Entity
public class Groupe {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(unique = true)
    private String groupName;
    @ManyToMany(fetch = FetchType.EAGER)
    private Collection<Utilisateur> utilisateurs=new ArrayList<>();
}
```

UTILISATEUR

- + ID
- + PASSWORD
- + USER_NAME

GROUPE_UTILISATEURS

- + GROUPES_ID
- + UTILISATEURS_ID

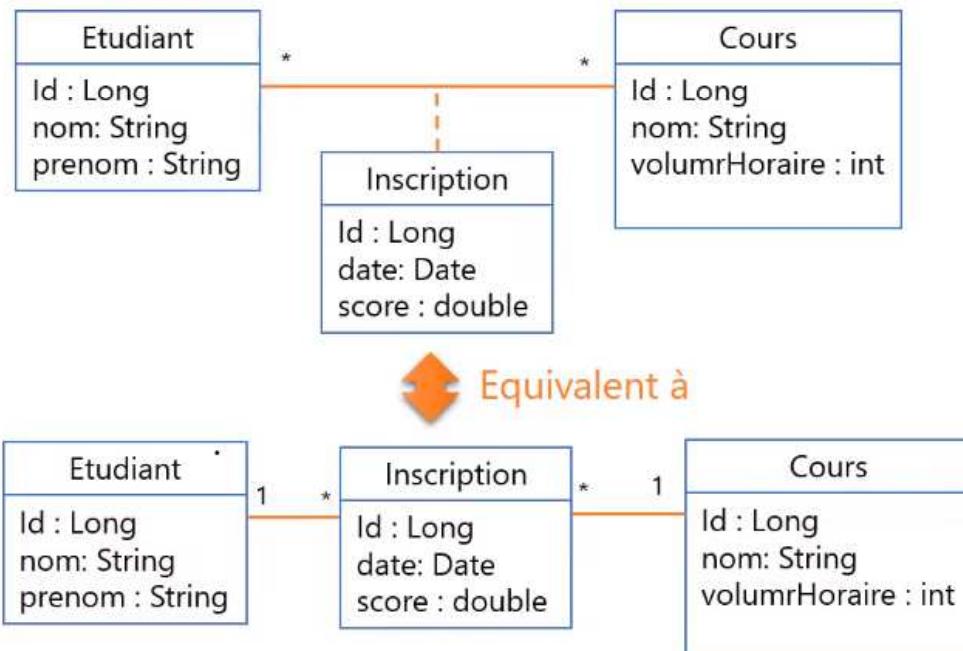
GROUPE

- + ID
- + GROUP_NAME

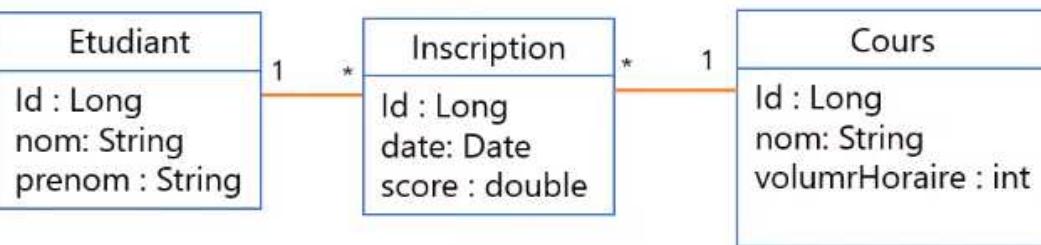
Cas de ManyMany avec Classe d'association

Exemple :

- Un Etudiant peut s'inscrire dans plusieurs Cours à une date donnée avec un score obtenu. Un cours concerne plusieurs Inscriptions. (Plusieurs à Plusieurs avec Classe d'association Inscription)
- Équivalent à : Un Etudiant peut effectuer Plusieurs Inscription. Chaque Inscription Concerne un Cours
- Deux Associations : Un à Plusieurs + Plusieurs à 1



Cas de ManyMany avec Classe d'association



```
@Entity
public class Etudiant {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private String prenom;
    @OneToMany(mappedBy = "etudiant")
    private Collection<Inscription> inscriptions;
}
```

```
@Entity
public class Cours {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private int volumeHoraire;
    @OneToMany(mappedBy = "cours")
    private Collection<Inscription> inscriptions;
}
```

```
@Entity
public class Inscription {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date dateInscription;
    private double score;
    @ManyToOne
    private Etudiant etudiant;
    @ManyToOne
    private Cours cours;
}
```

```
public interface CoursRepository extends JpaRepository<Cours,Long> {}
```

```
public interface EtudiantRepository extends JpaRepository<Etudiant,Long> {}
```

```
public interface InscriptionRepository extends JpaRepository<Inscription,Long> {
    List<Inscription> findByCours(Cours cours);
}
```

Mapping de l'héritage

- Différentes stratégies de Mapping de l'héritage
- Une table par hiérarchie ([SINGLE_TABLE](#))
- Une table pour chaque classe concrète ([TABLE_PER_CLASS](#))
- Une table pour la classe parente et une table pour chaque classe fille ([JOINED_TABLE](#))

Personne
Id : Long
Nom: String
dateNaissance : Date

SINGLE_TABLE : PERSONNES

ID	TYP E	NO M	DATE_NAISSAN CE	NOT E	MAIETRE
1	ET	N1	d1	15	NULL
2	PRO F	N2	d2	NULL	MATH

Etudiant
note: double

Enseignant
matiere: String

JOINED_TABLE

ETUDIANTS	PERSONNES
NOTE	#ID
15	1
ENSEIGNANTS	PERSONNES
MATIERE	#ID
MATH	2

TABLE_PER_CLASS

ETUDIANTS

ID	NO M	DATE_NAISSAN CE	NOTE
1	N1	d1	15

ENSEIGNANTS

ID	NO M	DATE_NAISSAN CE	MATIERE
2	N2	d2	MATH

Mapping de l'héritage : Stratégie SINGLE_TABLE

Personne
Id : Long
Nom: String
dateNaissance :

SELECT * FROM PERSONNE;

TYPE	ID	DATE_NAISSANCE	NOM	MATIERE	NOTE
ETUD	1	2020-09-03 20:05:28.358	Hassan	null	14.0
PROF	2	2020-09-03 20:05:28.362	Mohamed	MATH	null

Date



Etudiant
note: double

Enseignant
matiere: String

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "TYPE", length = 4)
public abstract class Personne {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private Date dateNaissance;
}
```

```
public interface PersonneRepository extends JpaRepository<Personne, Long> {}
```

```
@Entity
@DiscriminatorValue("ETUD")
public class Etudiant extends Personne {
    private double note;
}
```

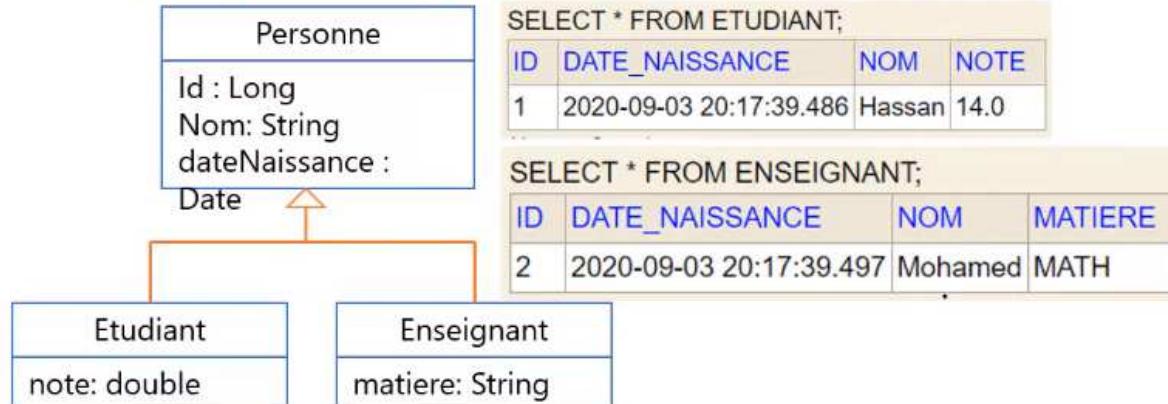
```
@Entity
@DiscriminatorValue("PROF")
public class Enseignant extends Personne {
    private String matiere;
}
```

```
@Autowired
private PersonneRepository personneRepository;
```

```
Etudiant etudiant=new Etudiant();
etudiant.setNom("Hassan");
etudiant.setDateNaissance(new Date());
etudiant.setNote(14);
personneRepository.save(etudiant);
```

```
Enseignant enseignant=new Enseignant();
enseignant.setNom("Mohamed");
enseignant.setDateNaissance(new Date());
enseignant.setMatiere("MATH");
personneRepository.save(enseignant);
```

Mapping de l'héritage : Stratégie TABLE_PER_CLASS



```

@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Personne {
    @Id @GeneratedValue(strategy = GenerationType.TABLE)
    private Long id;
    private String nom;
    private Date dateNaissance;
}
    
```

```

public interface PersonneRepository extends JpaRepository<Personne, Long> {
    
```

```

@Entity
public class Etudiant extends Personne {
    private double note;
}
    
```

```

@Entity
public class Enseignant extends Personne {
    private String matiere;
}
    
```

```

@Autowired
private PersonneRepository personneRepository;
    
```

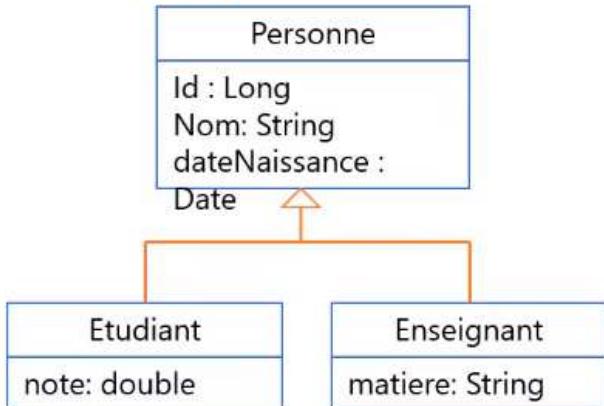
```

Etudiant etudiant=new Etudiant();
etudiant.setNom("Hassan");
etudiant.setDateNaissance(new Date());
etudiant.setNote(14);
personneRepository.save(etudiant);
    
```

```

Enseignant enseignant=new Enseignant();
enseignant.setNom("Mohamed");
enseignant.setDateNaissance(new Date());
enseignant.setMatiere("MATH");
personneRepository.save(enseignant);
    
```

Mapping de l'héritage : Stratégie JOINED_TABLE



SELECT * FROM PERSONNE;		
ID	DATE_NAISSANCE	NOM
1	2020-09-03 20:25:17.668	Hassan
2	2020-09-03 20:25:17.679	Mohamed

SELECT * FROM ETUDIANT;	
NOTE	ID
14.0	1

SELECT * FROM ENSEIGNANT;	
MATIERE	ID
MATH	2

```

@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Personne {
    @Id @GeneratedValue(strategy = GenerationType.TABLE)
    private Long id;
    private String nom;
    private Date dateNaissance;
}
  
```

```

public interface PersonneRepository extends JpaRepository<Personne,Long> {
}
  
```

```

@Entity
public class Etudiant extends Personne {
    private double note;
}
  
```

```

@Entity
public class Enseignant extends Personne {
    private String matiere;
}
  
```

```

@Autowired
private PersonneRepository personneRepository;
  
```

```

Etudiant etudiant=new Etudiant();
etudiant.setNom("Hassan");
etudiant.setDateNaissance(new Date());
etudiant.setNote(14);
personneRepository.save(etudiant);
  
```

```

Enseignant enseignant=new Enseignant();
enseignant.setNom("Mohamed");
enseignant.setDateNaissance(new Date());
enseignant.setMatiere("MATH");
personneRepository.save(enseignant);
  
```

Transformation des associations N à N

Contrainte de clef étrangère UN A PLUSIEURS avec @ManyToOne (meilleure méthode pour mapper une association 1 à plusieurs)



```
class Commande {  
    ...  
    @ManyToOne(fetch = FetchType.LAZY, optional = false)  
    @JoinColumn(name = "idclient")  
    @OnDelete(action = OnDeleteAction.NO_ACTION)  
    private Client client;  
}
```

- **FetchType.LAZY :**

L'attribut est chargé uniquement au moment de l'accès, et non lors du chargement initial de l'objet.

- **FetchType.EAGER :**

L'attribut est chargé initialement lors du chargement de l'objet.

- **optional = true :**

Décider si la multiplicité est 0..1 (optional = true) ou 1..1 (optional = false). Dans le cas de la valeur true, la colonne de la clef étrangère peut accepter la valeur null.

Effectuer des opérations de CRUD sur les objets d'une entité

- **mappedBy = "client"** : mappedBy est utilisé pour spécifier l'entité qui contrôle (owner) la relation un à plusieurs. Non nécessaire en cas d 'association 1 à plusieurs unidirectionnelle. Le nom spécifié dans mappedBy doit être celui que porte l'attribut « un » dans l'entité « plusieurs ».
- **cascade = CascadeType .ALL** : Définir l'action à entreprendre en cascade concernant les objets enfants en cas de manipulation du parent.
 - o **CascadeType.PERSIST** : sauvegarder (insérer) la liste des objets enfants si le parent est sauvegardé,
 - o **CascadeType.MERGE** : sauvegarder avec le même ID l'objet parent après des modifications, la liste des objets enfants est aussi sauvegardée,
 - o **CascadeType.DETACH** : DETACH a pour but de détacher l'objet entité ciblé du manager d 'entité qui le gère. Cette option est utilisée quand on a l'intention d 'exclure un objet entité d 'une opération (DELETE par exemple). Dans ce cas **CascadeType.DETACH** signifie que les objets enfants aussi seront détachés.
 - o **CascadeType.REFRESH** : REFRESH constitue l'inverse de MERGE, l'objet parent est restitué à son état initial après des modifications, ce qui veut dire que ces modifications sont annulées. Dans ce cas CascadeType.REFRESH signifie que les modifications des objets enfants aussi seront annulées.
 - o **CascadeType.ALL** : signifie le regroupement de toutes les options précédentes.
 - o Aucune action n'est prise par défaut , ce qui veut dire que si aucune option cascade n'est spécifiée , les objets enfants ne seront pas affectés par les actions subies par le parent.