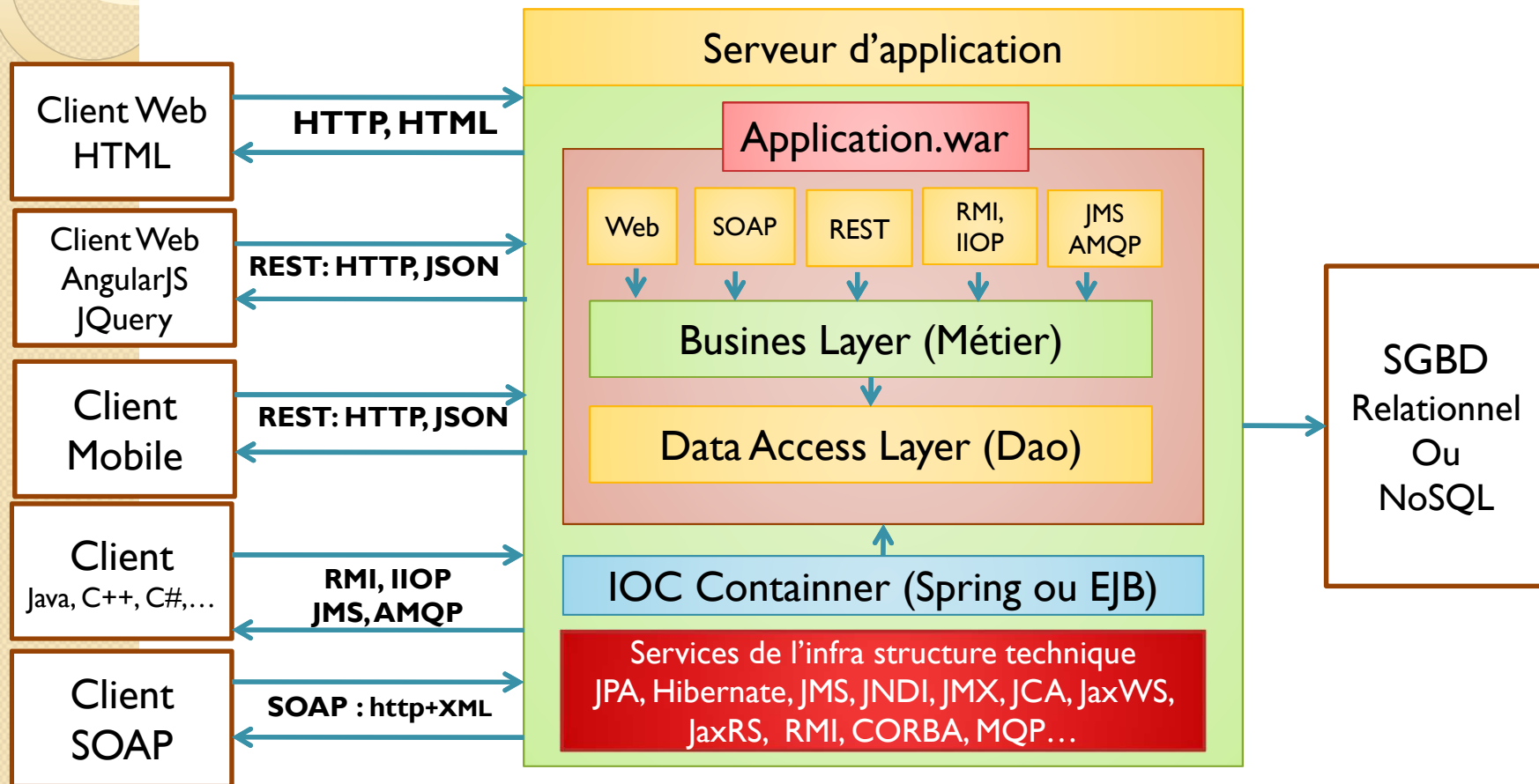




Java Persistence Api (JPA) Hibernate

Mohamed Youssfi : med@youssfi.net

Architecture JEE

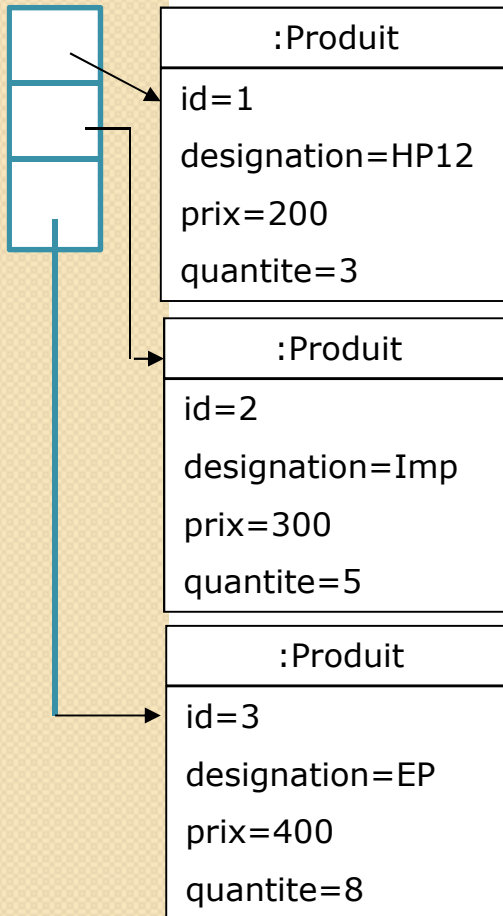




Mapping Objet Relationnel avec Hibernate

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

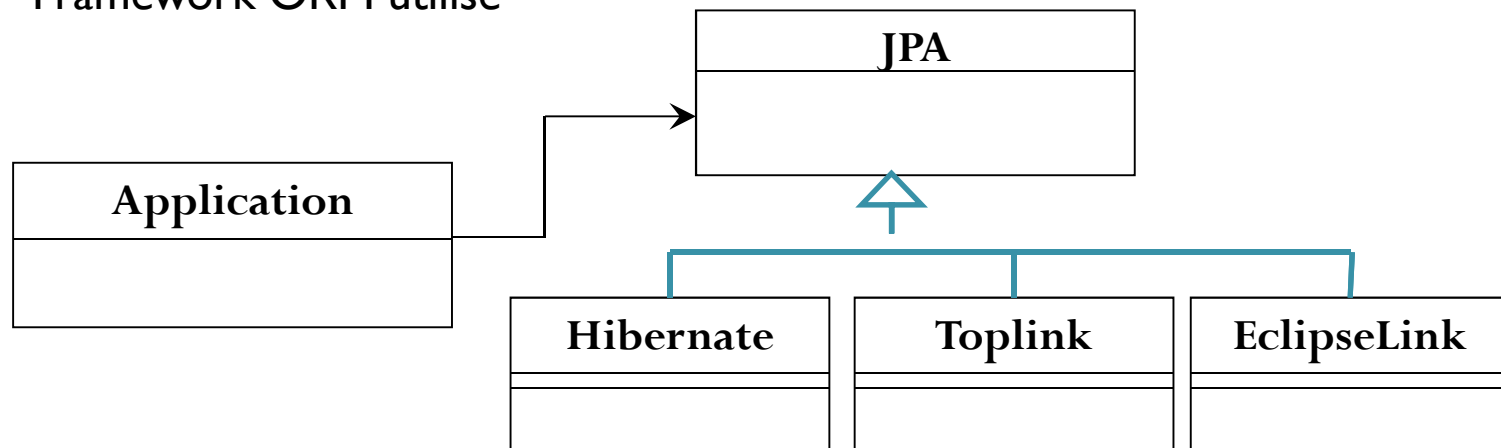
- Hibernate s'occupe du transfert des objets Java dans les tables de la base de données
- En plus, il permet de requêter les données et propose des moyens de les récupérer.
- Il peut donc réduire de manière significative le temps de développement qui aurait été autrement perdu dans une manipulation manuelle 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 changez de SGBD.
- Hibernate propose au développeur des méthodes d'accès aux bases de données plus efficaces ce qui devrait rassurer les développeurs.

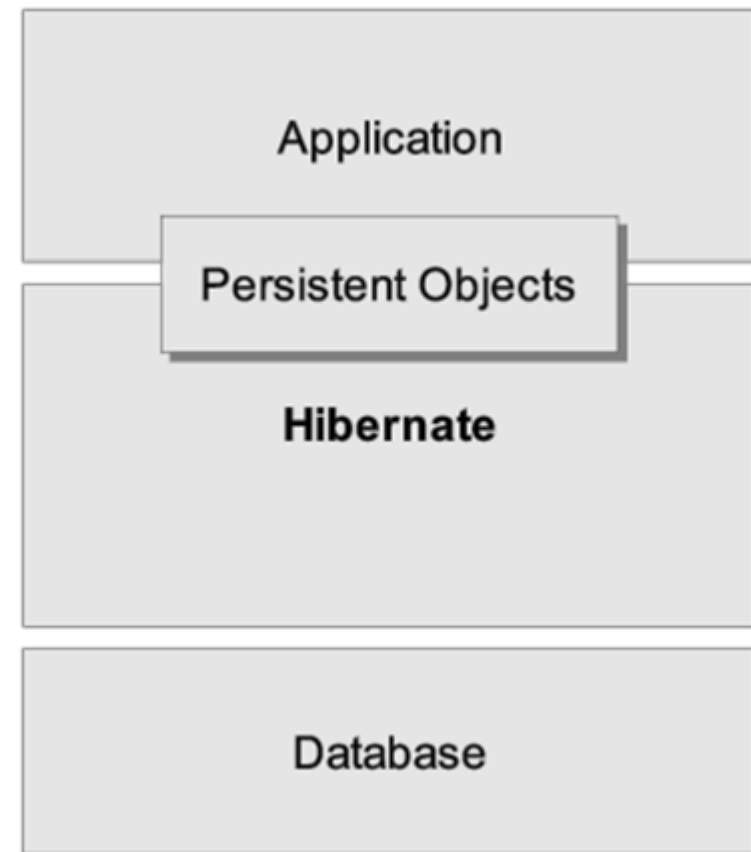
JPA : Java Persistence API

- 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 fichiers 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 ce cours, 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

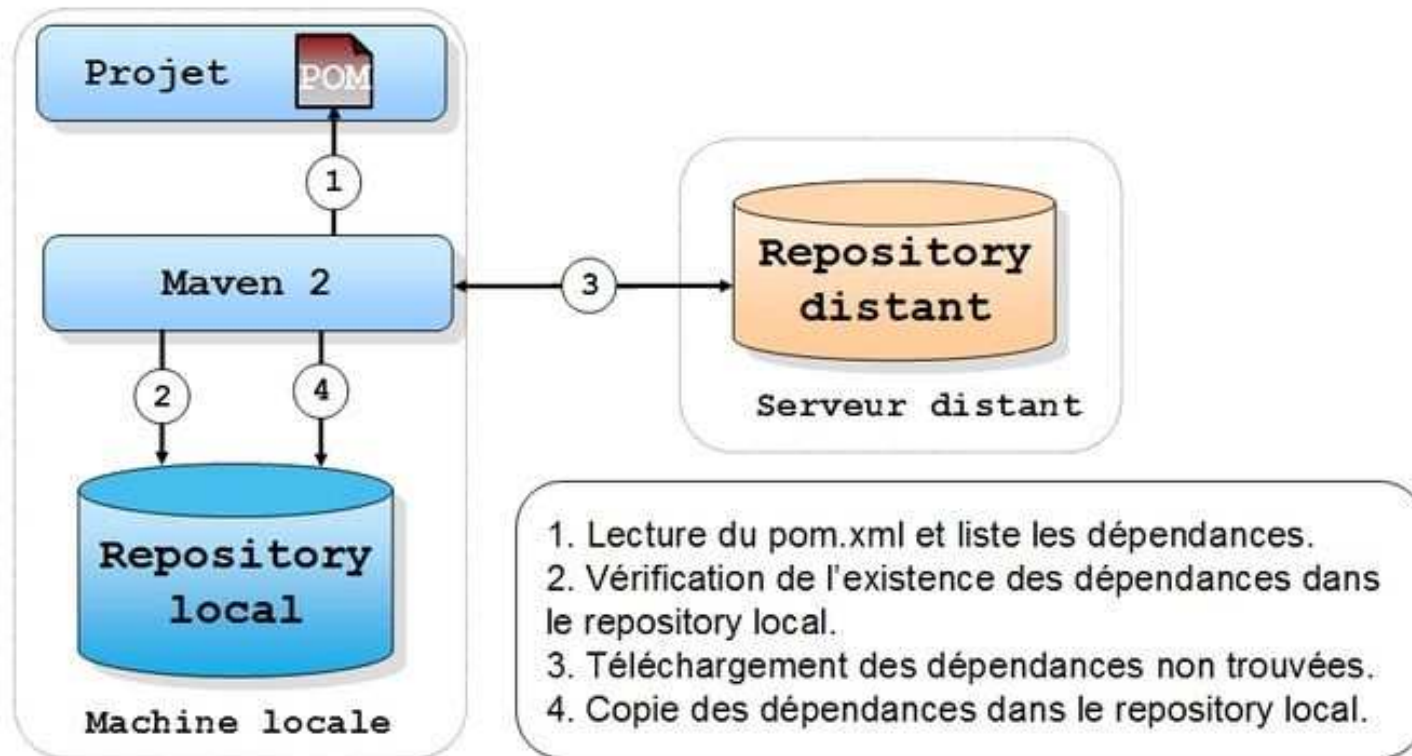
- ▲  TPH1
 - ▲  src/main/java
 - ▲  dao
 - ▷  CatalogueDaoImpl.java
 - ▷  ICatalogueDao.java
 - ▷  Produit.java
 - ▷  Test.java
 - ▲  src/main/resources
 - ▲  META-INF
 -  persistence.xml
 -  src/test/java
 -  src/test/resources
- ▷  JRE System Library [J2SE-1.5]
- ▷  Maven Dependencies
- ▷  src
-  target
-  pom.xml

Maven

- **Maven**, géré par l'organisation *Apache Software Foundation*. (*Jakarta Project*), est un **outil pour la gestion et l'automatisation de production des projets logiciels Java en général et Java EE en particulier**.
- L'objectif recherché est de
 - produire un logiciel à partir de ses sources,
 - en optimisant les tâches réalisées à cette fin
 - et en garantissant le bon ordre de fabrication.
 - **Compiler, Tester, Contrôler, produire les packages livrables**
 - **Publier la documentation et les rapports sur la qualité**
- **Apports :**
 - Simplification du processus de construction d'une application
 - Fournit les bonnes pratiques de développement
 - Tend à uniformiser le processus de construction logiciel
 - Vérifier la qualité du code
 - Faciliter la maintenance d'un projet

Maven et JUnit : Quelques Principes fondamentaux

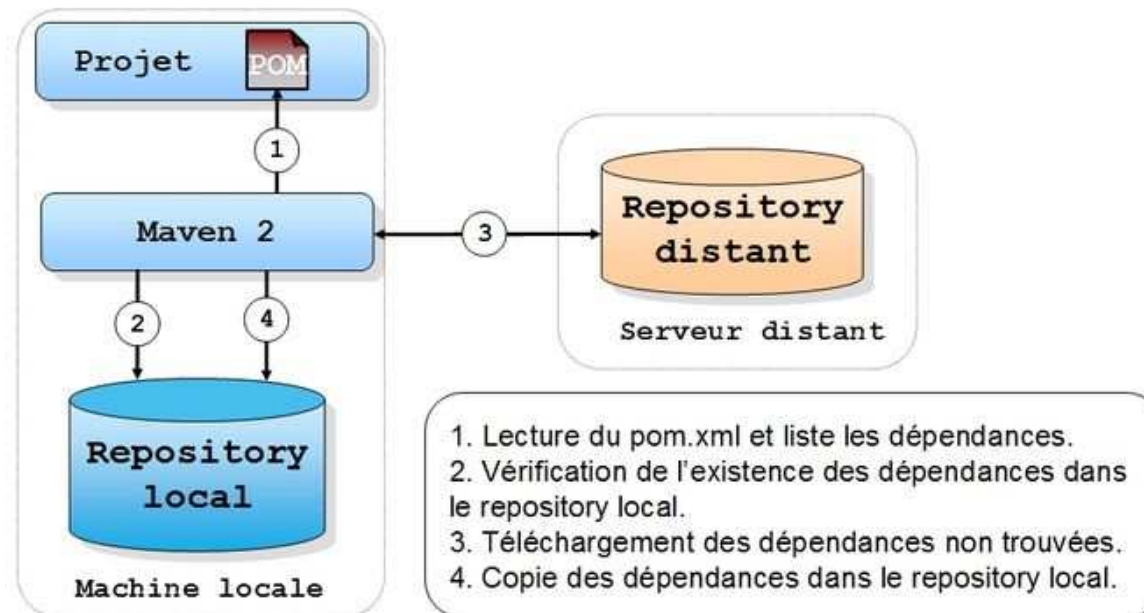
Gestion des dépendances par Maven 2



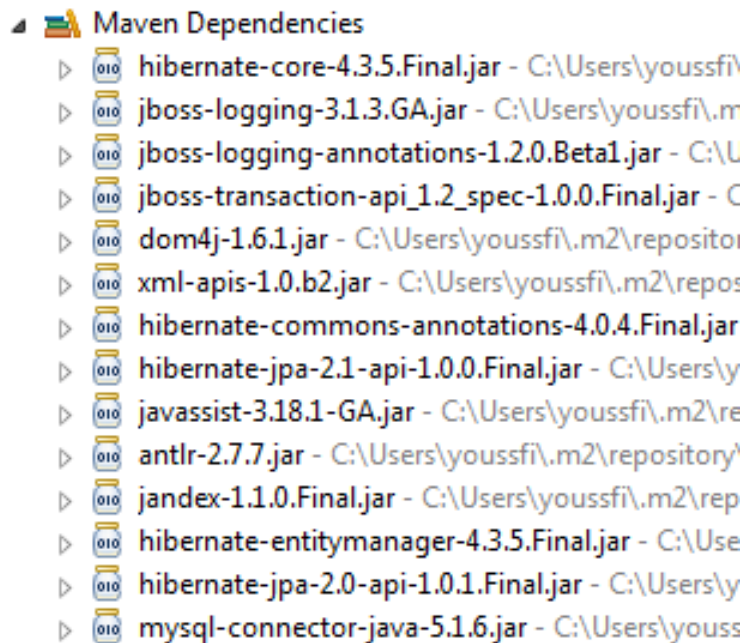
Maven : POM

- Maven utilise un paradigme connu sous le nom de **Project Object Model (POM)** afin de :
 - Décrire un projet logiciel,
 - Ses dépendances avec des modules externes
 - et l'ordre à suivre pour sa production.
- Il est livré avec un grand nombre de tâches (**GOLS**) prédéfinies, comme la compilation du code Java ou encore sa modularisation.

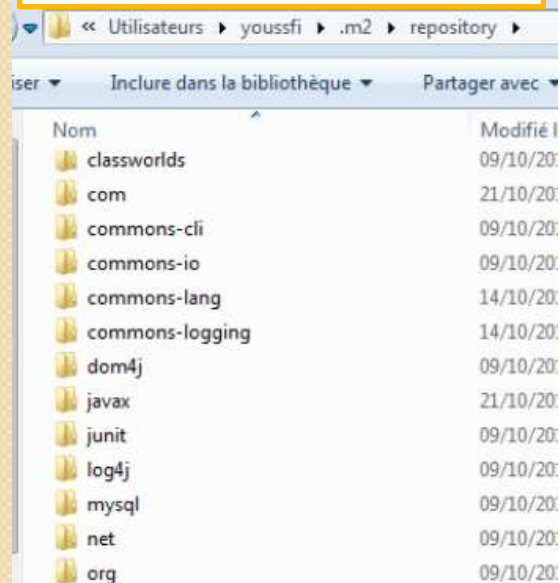
Gestion des dépendances par Maven 2



Dépendances maven



Local Maven Repository



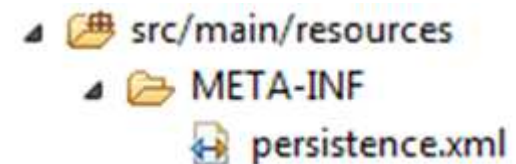
```
<dependencies>
<!-- Hibernate -->
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>4.3.5.Final</version>
  </dependency>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>4.3.5.Final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

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
}
```

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
<u>REF</u>	bigint(20)			Non	Aucune	AUTO_INCREMENT
<u>DES</u>	varchar(255) latin1_swedish_ci			Oui	NULL	
<u>prix</u>	double			Non	Aucune	
<u>quantite</u>	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 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 {  
        /* enregistrer 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();  
    }  
}
```


Avec Spring IOC

@Transactional

```
public void addProduit(Produit p) {  
    entityManager.persist(p);  
}
```

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 JPQL.
- 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 méthode 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

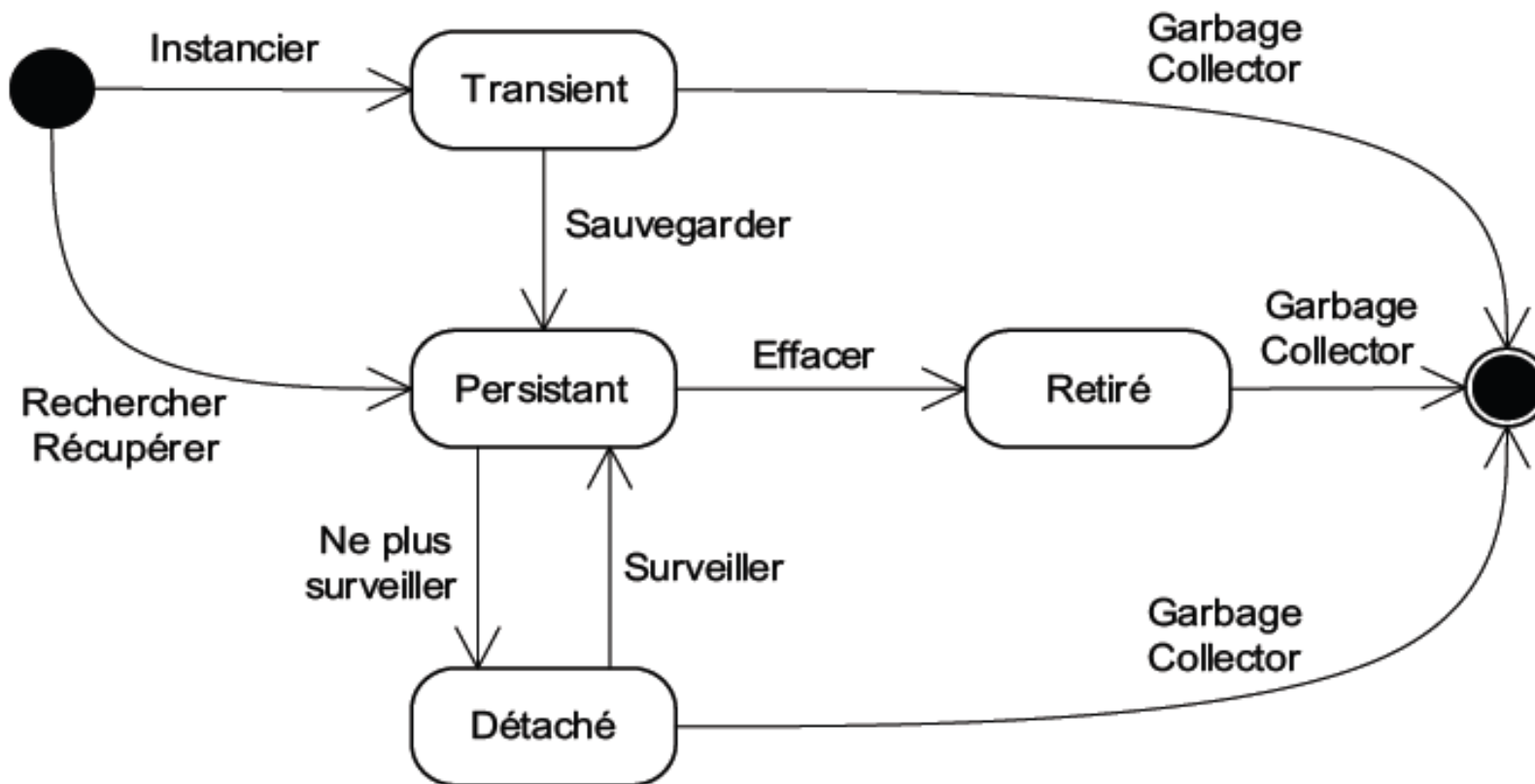
```
-----
Consulter les produits par mot clé
Hibernate: select produit0_.REF as REF1_0_, produit0_.DES as DES2_0_, produit0_.prix as prix3_0_,
produit0_.quantite as quantite4_0_ from PRODUITS produit0_ where produit0_.DES like ?
P1
P2
P3
```


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'une Entity JPA



Objet Persistant

- Un objet *persistant* 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.

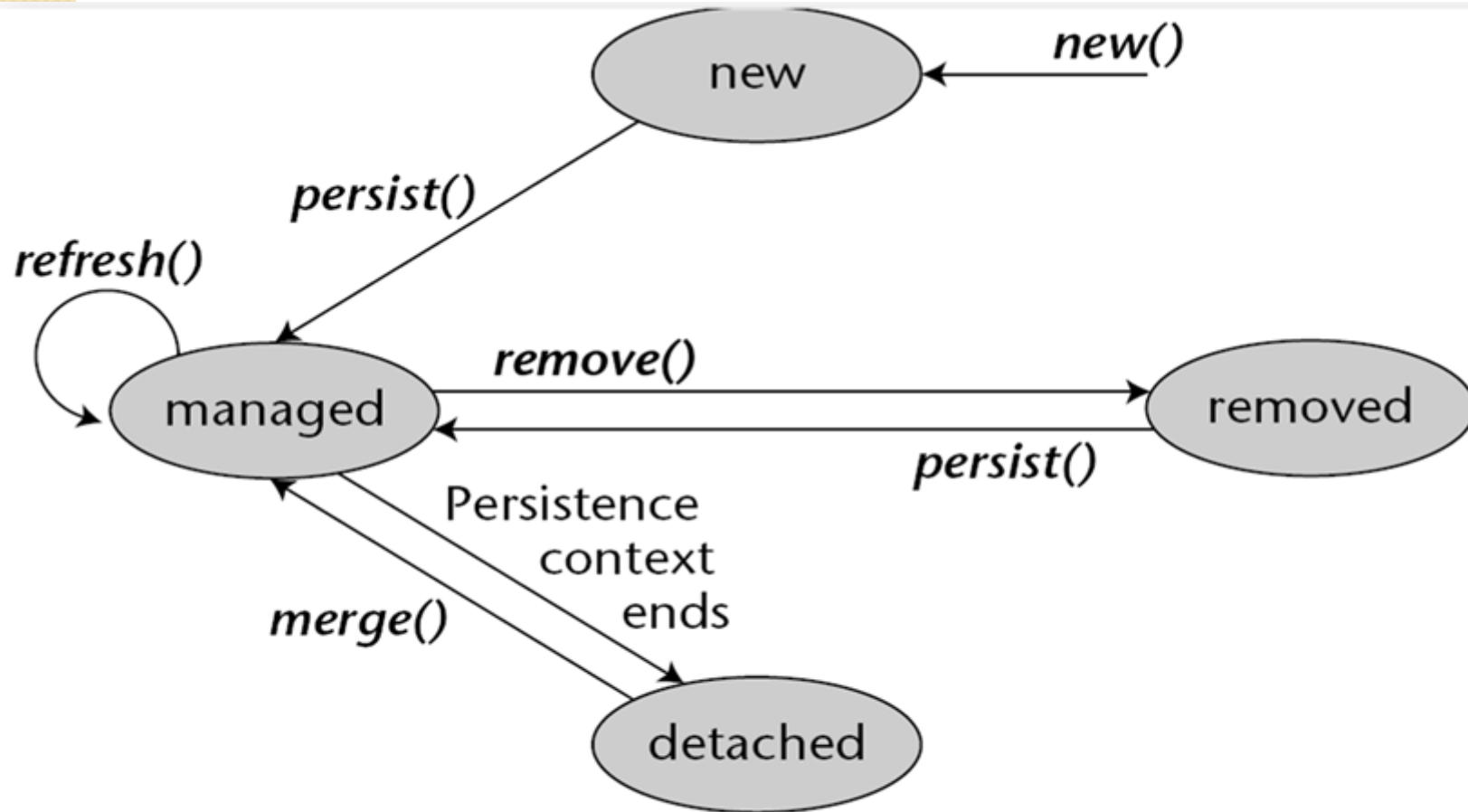
Etat Détaché

- 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



Entity life cycle.

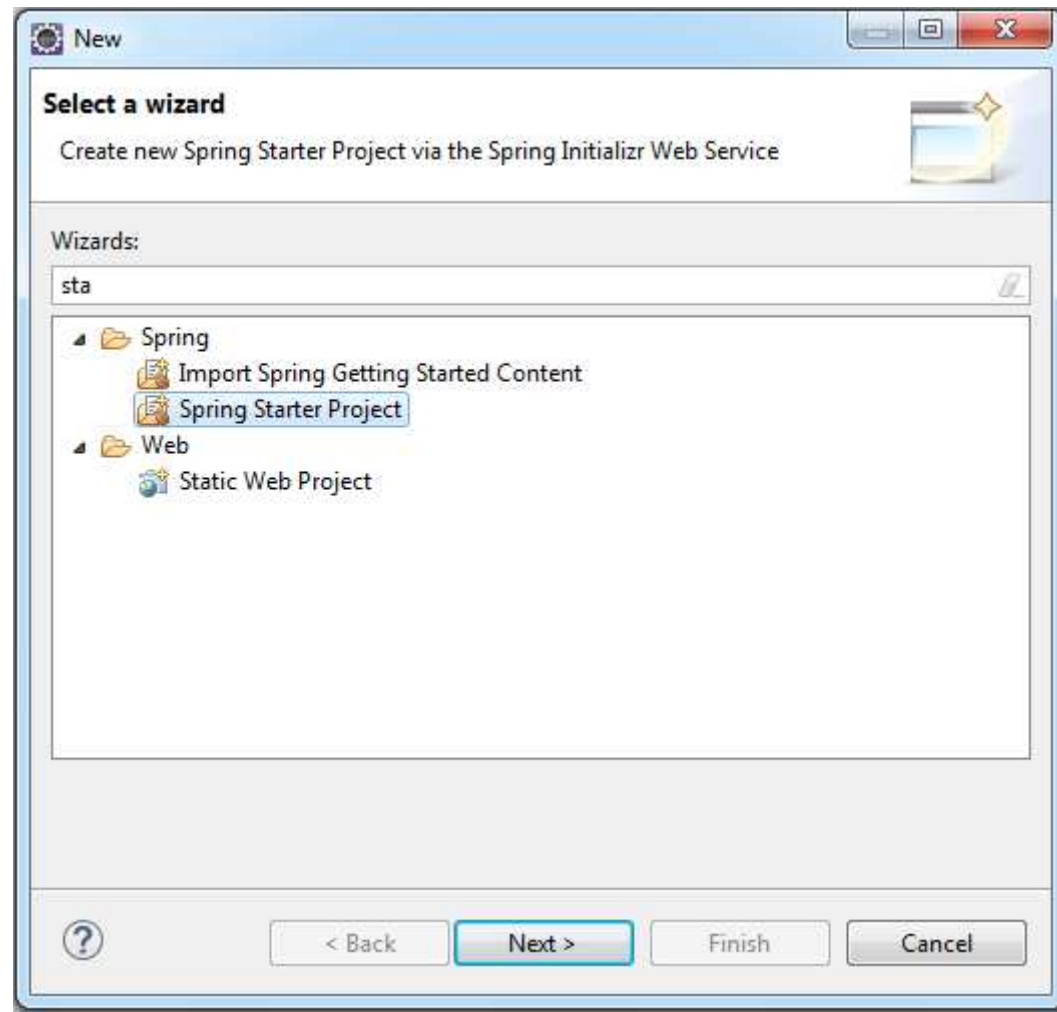


JPA DANS UN PROJET SPRING BOOT

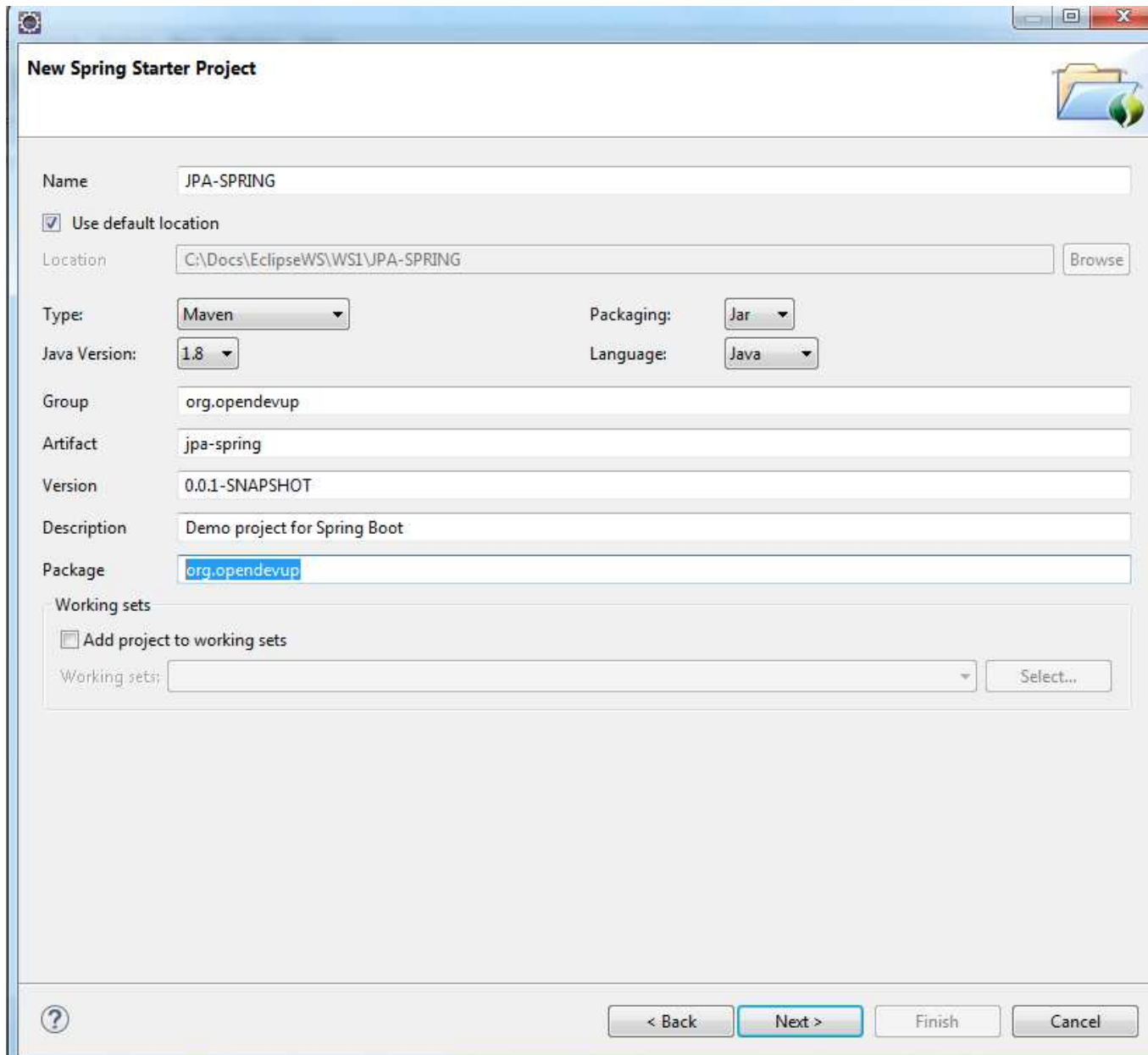
JPA dans un projet Spring

- 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

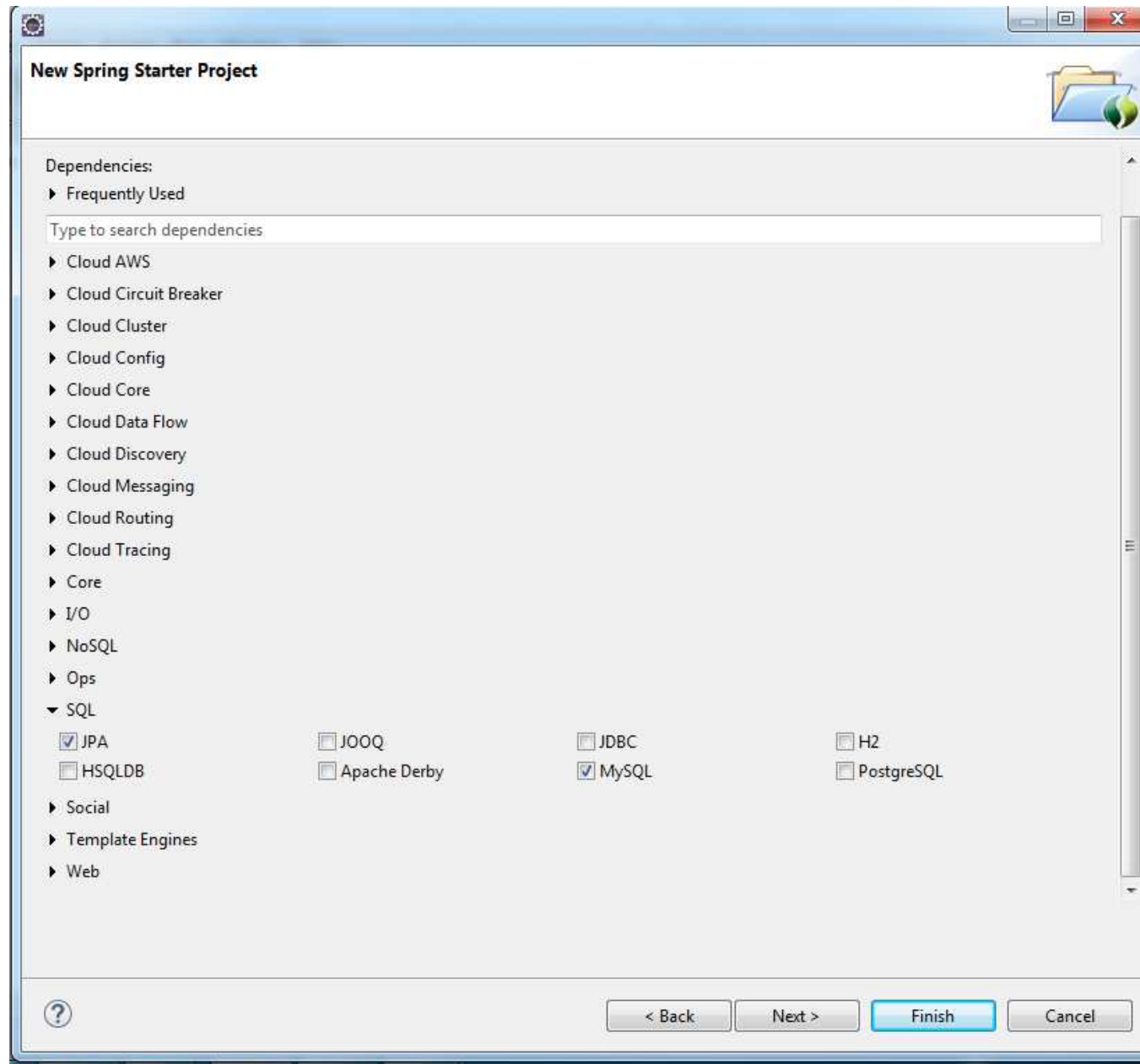


The screenshot shows the 'New Spring Starter Project' dialog box in the Eclipse IDE. The dialog is titled 'New Spring Starter Project' and contains the following fields and options:

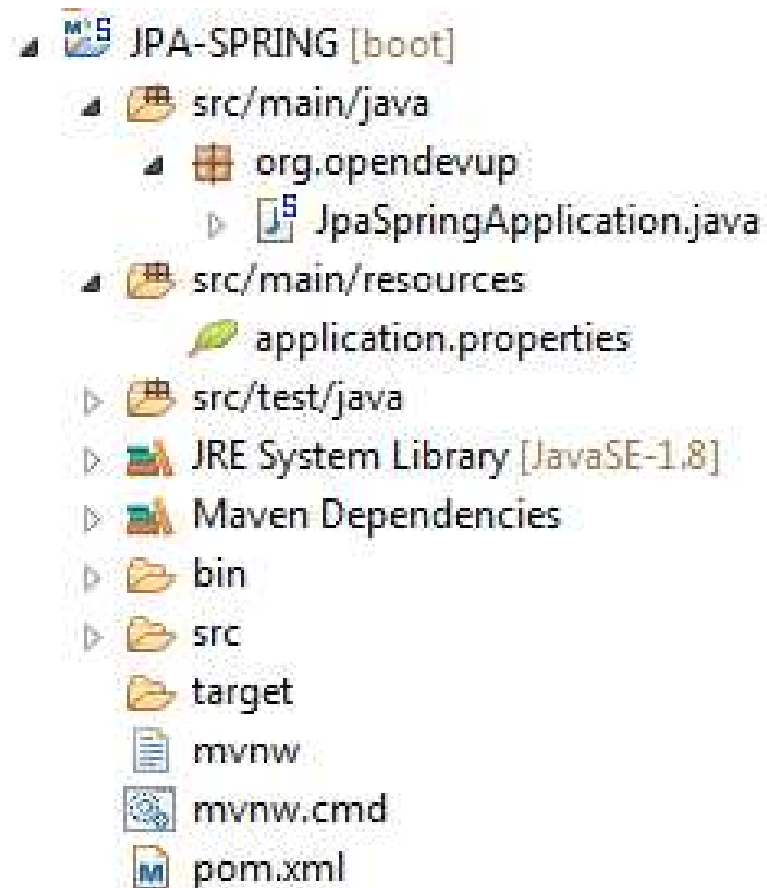
- Name:** JPA-SPRING
- ☒ **Use default location**
- Location:** C:\Docs\EclipseWS\WS1\JPA-SPRING (with a 'Browse' button)
- Type:** Maven (dropdown)
- Packaging:** Jar (dropdown)
- Java Version:** 1.8 (dropdown)
- Language:** Java (dropdown)
- Group:** org.opendevup
- Artifact:** jpa-spring
- Version:** 0.0.1-SNAPSHOT
- Description:** Demo project for Spring Boot
- Package:** org.opendevup
- Working sets:**
 - ☐ Add project to working sets
 - Working sets: (dropdown) (with a 'Select...' button)

At the bottom of the dialog, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted with a blue border.

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

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

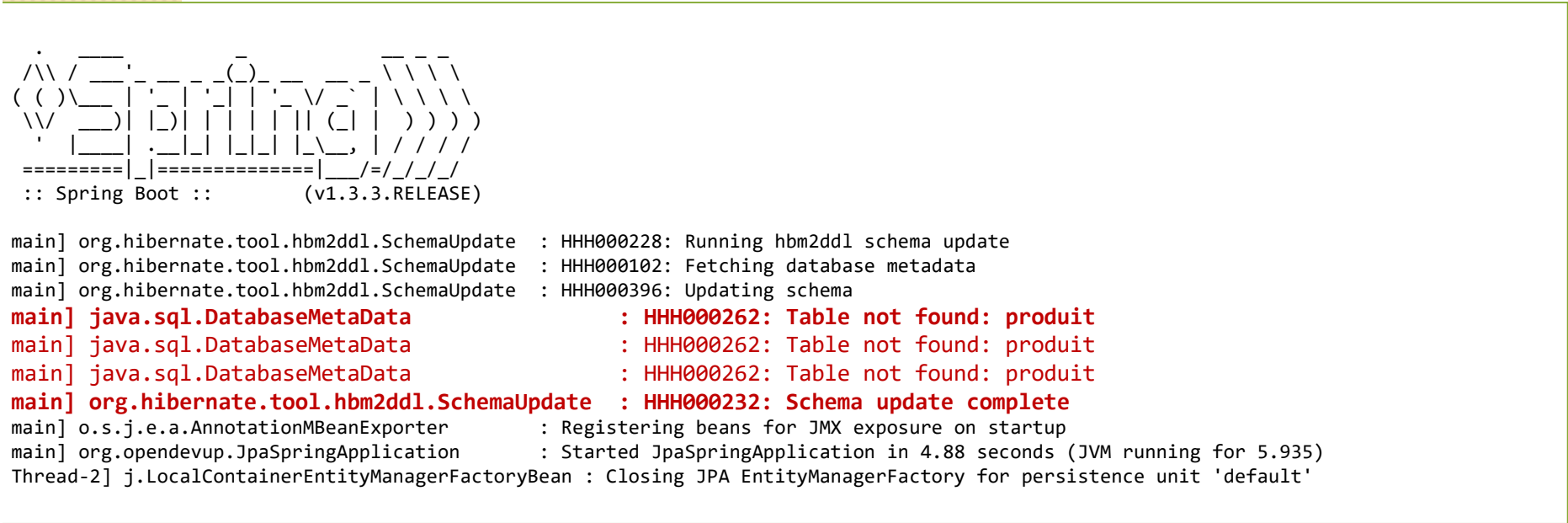
import java.io.Serializable;

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);
    }
}
```



Structure de la table Produits générée

	Colonne	Type	Interclassement	Attributs	Null	Défaut	Extra
<input type="checkbox"/>	id_produit	bigint(20)			Non	<i>Aucun</i>	AUTO_INCREMENT
<input type="checkbox"/>	designation	varchar(255)	latin1_swedish_ci		Oui	<i>NULL</i>	
<input type="checkbox"/>	prix	double			Non	<i>Aucun</i>	

Interface ICatalogueDao

```
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 CatalogueJpaImpl

```
@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 CatalogueJpaImpl

```
@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());  
}
```

Tester les méthodes

```
System.out.println("Supprimer un produit");
```

```
dao.remove(3L);
```

```
System.out.println("Afficher tous les produits ");
```

```
List<Produit> listProduits2=dao.findAll();
```

```
for(Produit pr:listProduits2){
```

```
    System.out.println(pr.getDesignation()+"---"+pr.getPrix());
```

```
}
```

```
}
```

```
}
```

Exécution

Instanciation de CatalogueDaoImpl

Ajouter des produits

Hibernate: insert into produit (designation, prix) values (?, ?)

Hibernate: insert into produit (designation, prix) values (?, ?)

Hibernate: insert into produit (designation, prix) values (?, ?)

Consulter tous les produits

Hibernate: select produit0_.id_produit as id_produ1_0_, produit0_.designation as designat2_0_, produit0_.prix as prix3_0_ from produit produit0_

AX 453----870.0

TA 6753----1230.0

HP 153----340.0

Consulter Un produit

Hibernate: select produit0_.id_produit as id_produ1_0_0_, produit0_.designation as designat2_0_0_, produit0_.prix as prix3_0_0_ from produit produit0_ where produit0_.id_produit=?

Produit :TA 6753

Mettre à jour le produit 2

Hibernate: select produit0_.id_produit as id_produ1_0_0_, produit0_.designation as designat2_0_0_, produit0_.prix as prix3_0_0_ from produit produit0_ where produit0_.id_produit=?

Hibernate: update produit set designation=?, prix=? where id_produit=?

Afficher les produits dont la désignation contient A

Hibernate: select produit0_.id_produit as id_produ1_0_, produit0_.designation as designat2_0_, produit0_.prix as prix3_0_ from produit produit0_ where produit0_.designation like ?

AX 453----870.0

Epson A76500----1230.0

Supprimer un produit

Hibernate: select produit0_.id_produit as id_produ1_0_0_, produit0_.designation as designat2_0_0_, produit0_.prix as prix3_0_0_ from produit produit0_ where produit0_.id_produit=?

Hibernate: delete from produit where id_produit=?





Afficher tous les produits

Hibernate: select produit0_.id_produit as id_produ1_0_, produit0_.designation as designat2_0_, produit0_.prix as prix3_0_ from produit produit0_

AX 453----870.0

Epson A76500----1230.0

Table Produits

T→		id_produit	designation	prix
		1	AX 453	870
		2	Epson A76500	1230



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 persistance. 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> {
    public List<Produit> findByDesignation(String des);
}
```

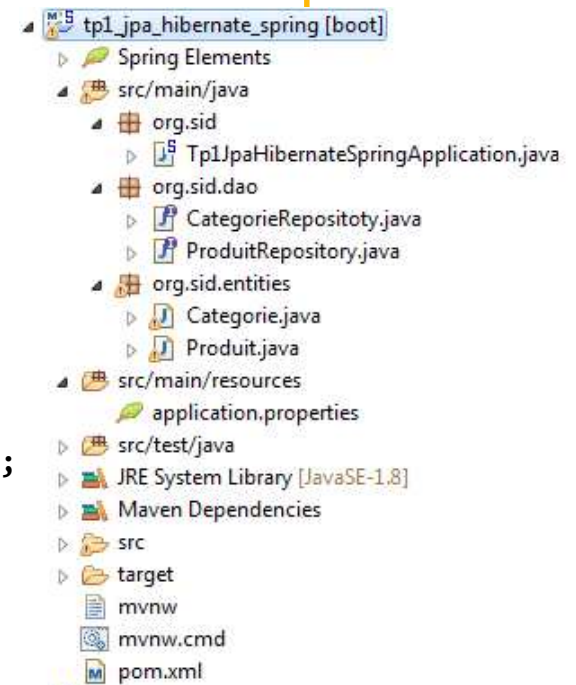
Exemple d'interface JpaRepository

```
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> {
    @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);  
        CategorieRepository categorieRepository=  
ctx.getBean(CategorieRepository.class);  
        categorieRepository.save(new Categorie("Ordinateurs"));  
        categorieRepository.save(new Categorie("Imprimantes"));  
        categorieRepository.save(new Categorie("Logiciels"));  
        ProduitRepository produitRepository=ctx.getBean(ProduitRepository.class);  
        Categorie c1=categorieRepository.findOne(1L);  
        Categorie c2=categorieRepository.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()); }  
    }  
}
```





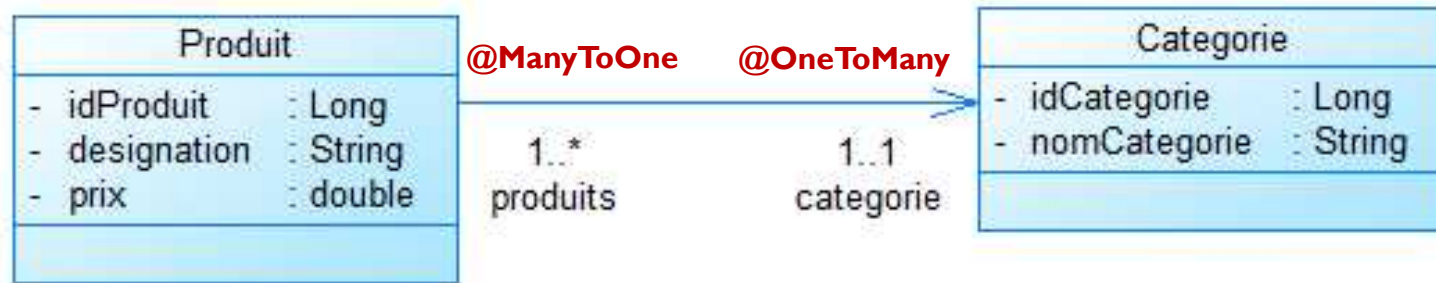
CAS D'UNE ASSOCIATION DE TYPE ONE TO MANY

Exemple d'application

- Supposant que l'on souhaite créer une application qui permet de gérer le catalogue des produits appartenant à des catégories.
- Chaque produit est défini par :
 - Son id de type Long (Auto Increment)
 - Sa désignation de type String
 - Son prix de type double
- Une catégorie est définie par :
 - Son code de type Long (Auto Increment)
 - Son nom de type String
- L'application doit permettre
 - D'ajouter une nouvelle catégorie
 - Ajouter un produit appartenant à une catégorie
 - Consulter toutes les catégories
 - Consulter les produits dont le nom contient un mot clé
 - Consulter les produits d'une catégorie
 - Consulter un produit
 - Mettre à jour un produit
 - Supprimer une catégorie
- L'injection des dépendances sera assurée par Spring IOC

Entités : Produit et Categorie

Diagramme de classes : Modèle Objet

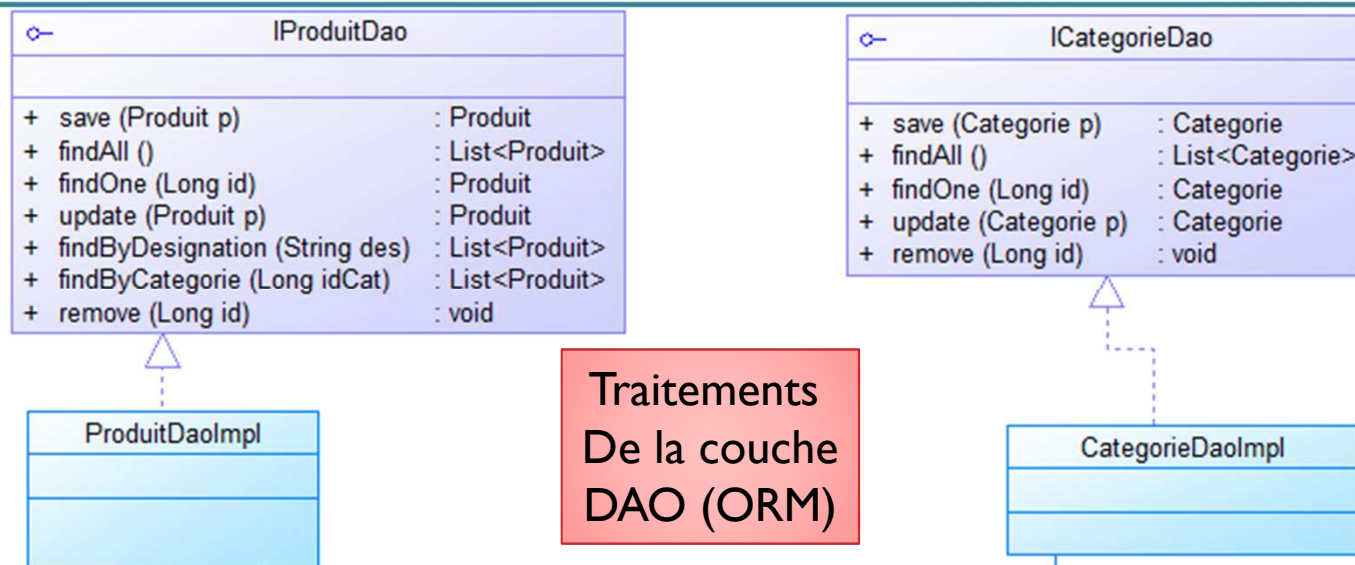
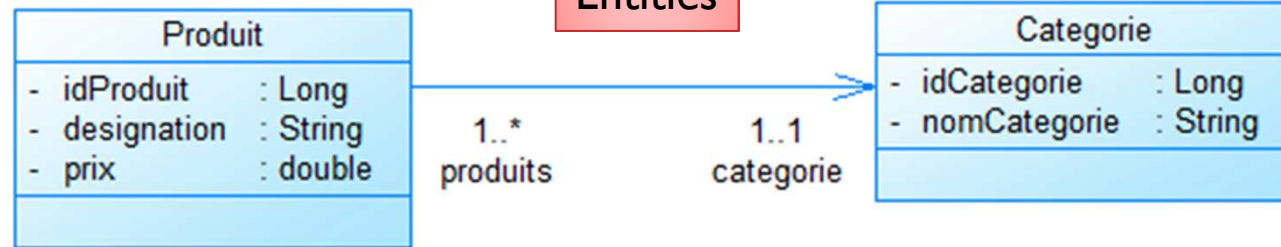


MLDR : Modèle Relationnel

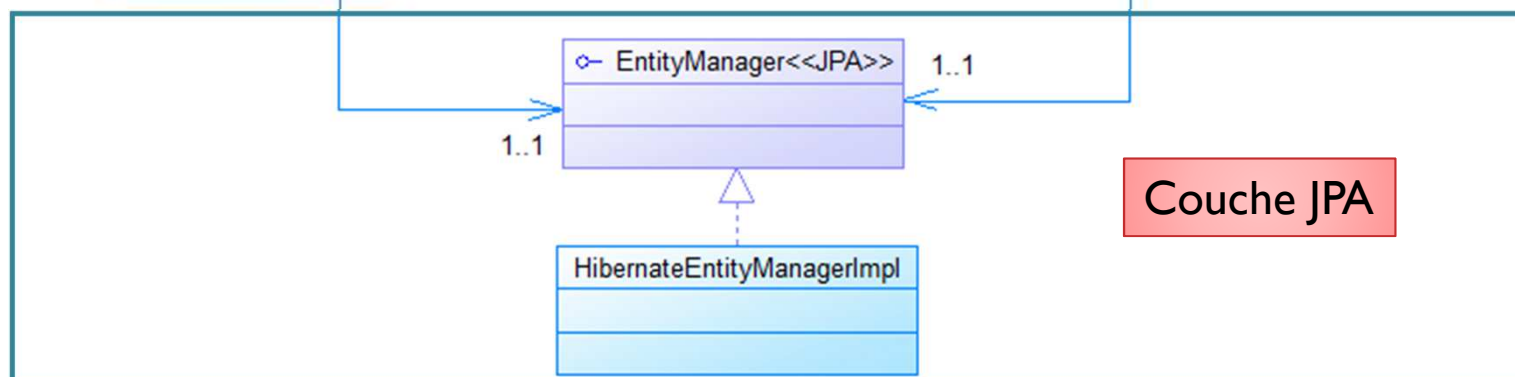
- CATEGORIES (ID_CAT, NOM_CAT)
- PRODUITS (ID_PRODUIT, DESIGNATION, PRIX, #ID_CAT)

Couche DAO

Entities

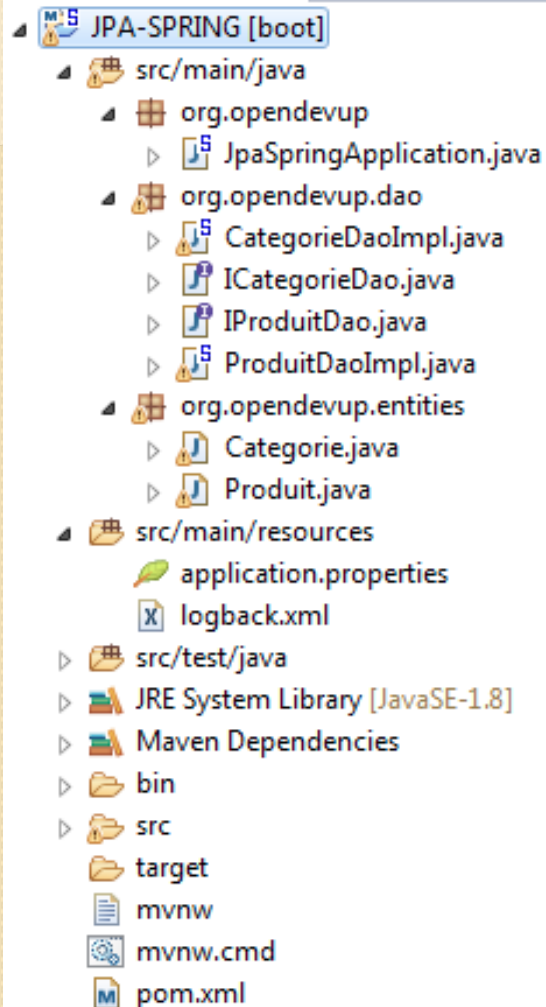


Traitements De la couche DAO (ORM)



Couche JPA

Structure du projet



logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

</configuration>
```

application.properties

```
spring.datasource.url = jdbc:mysql://localhost:3306/db_cat_Prod
spring.datasource.username = root
spring.datasource.password =
spring.datasource.driverClassName = com.mysql.jdbc.Driver
spring.jpa.show-sql = false
#spring.jpa.hibernate.ddl-auto = update
spring.jpa.hibernate.ddl-auto = create
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
spring.main.banner-mode=off
```


Entité : Produit

```
package dao;

import java.io.Serializable; import javax.persistence.*;

@Entity
public class Produit implements Serializable {
    @Id @GeneratedValue
    private Long idProduit;
    private String designation; private double prix;
    @ManyToOne
    @JoinColumn(name="ID_CAT")
    private Categorie categorie;
    public Produit() { }

    public Produit(String designation, double prix, Categorie categorie) {
        this.designation = designation;
        this.prix = prix;
        this.categorie = categorie;
    }

    // Getters et Setters
}
```

Entité : Catégorie

```
package dao;
import java.io.Serializable; import java.util.Collection;
import javax.persistence.*;
@Entity
public class Catégorie implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long idCatégorie;
    private String nomCatégorie;
    @OneToMany(mappedBy="catégorie", fetch=FetchType.LAZY)
    private Collection<Produit> produits;
    public Catégorie() { }
    public Catégorie(String nomCatégorie) {
        this.nomCatégorie = nomCatégorie;
    }
    public Catégorie(Long idCatégorie) {this.idCatégorie = idCatégorie;}
    // Getters et Setters
}
```


Interface ProduitDao

```
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);
    public List<Produit> findByCategorie(Long idCat);
}
```

Implémentation JPA de IProduitDao

```
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;
    }
    @Override
    public List<Produit> findAll() {
        Query req=entityManager.createQuery("select p from Produit p");
        return req.getResultList();
    }
}
```

Implémentation JPA de IProduitDao

```
@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);
}

@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();
}
```

Implémentation JPA de IProduitDao

@Override

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

@Override

```
public List<Produit> findByCategorie(Long idCat) {  
    Query req=entityManager.createQuery("select p from Produit p where  
p.categorie.idCategorie=:x");  
    req.setParameter("x", idCat);  
    return req.getResultList();  
}  
}
```

Interface ICategorieDao

```
package org.opendevup.dao;  
import java.util.List;  
import org.opendevup.entities.Categorie;  
public interface ICategorieDao {  
    public Categorie save(Categorie c);  
    public List<Categorie> findAll();  
    public Categorie findOne(Long id);  
    public void remove(Long id);  
    public void update(Categorie c);  
}
```

Implémentation JPA de ICategorieDao

```
package org.opendevup.dao;

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

@Repository
@Transactional
public class CategorieDaoImpl implements ICategorieDao{
    @PersistenceContext
    private EntityManager entityManager;

    @Override
    public Categorie save(Categorie c) {
        entityManager.persist(c);
        return c;
    }
}
```

Implémentation JPA de ICategorieDao

```
@Override
public List<Categorie> findAll() {
    Query req=entityManager.createQuery("select c from Categorie c");
    return req.getResultList();
}

@Override
public Categorie findOne(Long id) {
    Categorie c=entityManager.find(Categorie.class,id);
    return c;
}

@Override
public void remove(Long id) {
    Categorie c=findOne(id);
    entityManager.remove(c);
}

@Override
public void update(Categorie c) {
    entityManager.merge(c);
}
}
```

Application

```
package org.opendevup;
import java.util.List; import org.opendevup.dao.ICategorieDao;
import org.opendevup.dao.IProduitDao; import org.opendevup.entities.Categorie;
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 produitDao=ctx.getBean(IProduitDao.class);
        ICategorieDao categorieDao=ctx.getBean(ICategorieDao.class);
        System.out.println("Ajouter des catégories");
        categorieDao.save(new Categorie("Ordinateurs")); categorieDao.save(new Categorie("Imprimantes"));
        categorieDao.save(new Categorie("Logiciels"));
        System.out.println("Ajouter des produits");
        produitDao.save(new Produit("Ord AX 453", 870,new Categorie(1L)));
        produitDao.save(new Produit("Ord TA 6753", 1230,new Categorie(1L)));
        produitDao.save(new Produit("Imp HP 153", 340,new Categorie(2L)));
        produitDao.save(new Produit("Camtasia Studio", 340,new Categorie(3L)));
    }
}
```


Application

```
System.out.println("Liste des toutes les catégories:");
categorieDao.findAll().forEach(c->System.out.println(c.getNomCategorie()));
System.out.println("Liste des tous les produits:");
produitDao.findAll().forEach(p->{
System.out.println(p.getIdProduit()+"--"+p.getDesignation()+"--"+p.getPrix()+"--
"+p.getCategorie().getNomCategorie());
});

System.out.println("Consulter Une catégorie :");
Categorie c=categorieDao.findOne(1L);
System.out.println("Nom Catégorie :"+c.getNomCategorie());

/*c.getProduits().forEach(p->System.out.println(p.getDesignation()));*/
System.out.println("Consulter les produits d'une catégorie :");
List<Produit> prodsParCat=produitDao.findByCategorie(1L);
prodsParCat.forEach(p->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

Application

- On souhaite créer une application qui permet de gérer des comptes bancaire.
 - Chaque compte est défini un code de type String, un solde et une date de création
 - Un compte courant est un compte qui possède en plus un découvert
 - Un compte épargne est un compte qui possède en plus un taux d'intérêt.
 - Chaque compte appartient à un client.
 - Chaque client est défini par son code et son nom
 - Chaque compte peut subir plusieurs opérations.
 - Il existe deux types d'opérations :Versement et Retrait
 - Une opération est définie par un numéro, une date et un montant.

Exigences fonctionnelles

- L'application doit permettre de :
 - Gérer des les clients :
 - Ajouter un client
 - Consulter tous les clients
 - Consulter les clients dont le nom contient un mot clé.
 - Gérer les comptes :
 - Ajouter un compte
 - Consulter un compte
 - Gérer les opérations :
 - Effectuer un versement d'un montant dans un compte
 - Effectuer un retrait d'un montant dans un compte
 - Effectuer un virement d'un montant d'un compte vers un autre
 - Consulter les opérations d'un compte page par page
- Les opérations nécessitent une opération d'authentification

Exigences Techniques

- Les données sont stockées dans une base de données MySQL
- L'application se compose de trois couches :
 - La couche DAO qui est basée sur Spring Data, JPA, Hibernate et JDBC.
 - La couche Métier
 - La couche Web basée sur MVC coté Serveur en utilisant Thymeleaf.
- La sécurité est basée sur Spring Security

Travail demandé :

- Etablir une architecture technique du projet
- Etablir un diagramme de classes qui montre les entités, la couche DAO et la couche métier.
- Créer un projet SpringBoot qui contient les éléments suivants :
 - Les entités
 - La couche DAO (Interfaces Spring data)
 - La couche métier (Interfaces et implémentations)
 - La couche web :
 - Les contrôleurs Spring MVC
 - Les Vue basée sur Thymeleaf
- Sécuriser l'application en utilisant un système d'authentification basé sur Spring Security

Architecture Technique

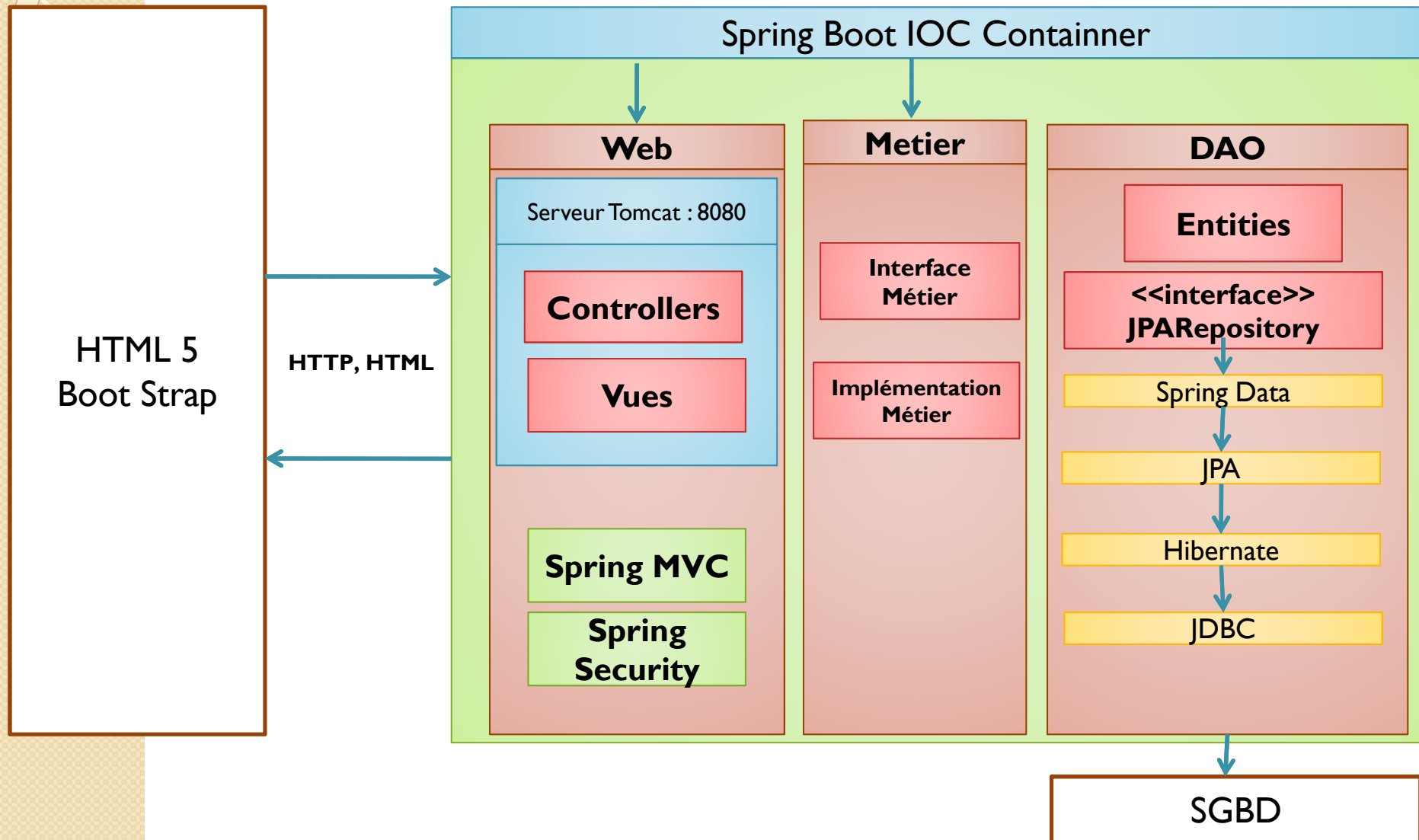
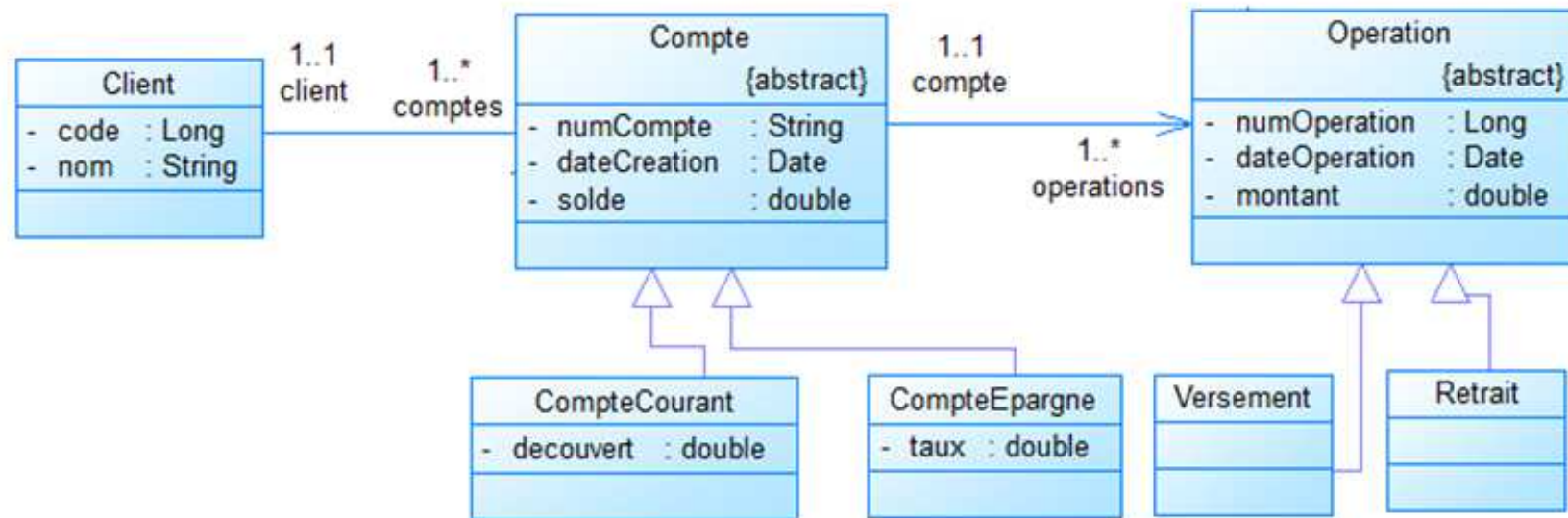


Diagramme de classes des entités et MLDR



- **MLRD** : En utilisant la stratégie **Single Table** pour l'héritage
 - **T_CLIENTS** (**CODE CLI**, NOM_CLI)
 - **T_COMPTE** (**NUM CPTE**, **TYPE PTE**, DATE_CR, SOLDE, DEC, TAUX, **#CODE_CLI**)
 - **T_OPERATIONS** (**NUM OP**, **TYPE OP**, DATE_OP, MONTANT, **#NUM_CPTE**)

Aperçu d'une vue de l'application web :

The screenshot shows a web browser window with the address bar set to `localhost:8080`. The application interface is divided into several sections:

- Compte:** A section for account management with a text input for "Numéro de compte" containing "CE1" and an "OK" button.
- Opérations sur le compte:** A section for account operations with radio buttons for "Versment" and "Retrait" (selected). Below is a "Montant:" input field with the value "10" and an "Effectuer" button.
- Solde:** Displays "Solde: 11973".
- Date Création:** Displays "Date Création :".
- Taux:** Displays "Taux: 2.5".
- Table of Transactions:** A table with columns "Numéro", "Date", "Montant", and "Type". It contains three rows of data.
- Navigation:** A row of buttons numbered 0 through 10, with button 0 highlighted.

Numéro	Date	Montant	Type
15	2015-11-24 08:34:57	0	V
16	2015-11-24 08:35:21	0	R
17	2015-11-24 08:44:18	0	V



COUCHE DAO

Entité JPA Client

```
@Entity
public class Client implements Serializable{
    @Id @GeneratedValue
    private Long code;
    private String nom;

    @OneToMany(mappedBy="client", fetch=FetchType
    .LAZY)
    private Collection<Compte> comptes;

    // Générer un constructeur sans param
    public Client() { super(); }
    // Générer un constructeur avec params
    public Client(String nom) { super();
    this.nom = nom;
    }
    //Générer les getters et setters
}
```

Entité JPA Compte

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn( name="TYPE_CPTE",
discriminatorType=DiscriminatorType.STRING,length=2)
public abstract class Compte implements Serializable {
    @Id
    private String code;
    private double solde;
    private Date dateCreation;
    @ManyToOne
    @JoinColumn(name="CODE_CLI")
    private Client client;
    @OneToMany(mappedBy="compte")
    private Collection<Operation> operations;

    // Générer un constructeur sans param
    public Compte() { super(); }
    // Générer un constructeur avec params
    public Compte(String code, double solde, Date
    dateCreation, Client client) {
        super(); this.code = code; this.solde = solde;
        this.dateCreation = dateCreation; this.client = client;
    }
    //Générer les getters et setters
}
```

Entité JPA CompteCourant

```
@Entity
@DiscriminatorValue("CC")
public class CompteCourant extends Compte {
    private double decouvert;

    // Constructeur sans param
    public CompteCourant() { super(); }
    // Constructeur avec params
    public CompteCourant(String code, double
solde, Date dateCreation, Client client,
double decouvert ) {
        super(code, solde, dateCreation, client);
        this.decouvert = decouvert;
    }

    // Getters et Setters

}
```

Entité JPA CompteEpargne

```
@Entity
@DiscriminatorValue("CE")
public class CompteEpargne extends Compte {
    private double taux;

    // Constructeur sans param
    public CompteEpargne() { super(); }
    // Constructeur avec params
    public CompteEpargne (String code, double solde, Date
dateCreation, Client client, double taux) {
        super(code, solde, dateCreation, client);
        this.taux = taux;
    }

    // Getters et Setters

}
```

Entité Operation

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="TYPE",length=2)
public abstract class Operation {
    @Id @GeneratedValue
    private Long numero;
    private Date dateOperation;
    private double montant;
    @ManyToOne
    @JoinColumn(name="CODE_CPTE")
    private Compte compte;
    // Constructeur sans param
    public Operation() { super(); }
    // Constructeur avec params
    public Operation(Date dateOperation, double montant,
    Compte compte) {
        super();
        this.dateOperation = dateOperation;
        this.montant = montant;
        this.compte = compte;
    }
    // Getters et Setters
}
```

Entité Versement

```
@Entity
@DiscriminatorValue("V")
public class Versement extends Operation{
    public Versement() { super();}
    public Versement(Date dateOperation,
    double montant, Compte compte) {
        super(dateOperation, montant, compte);
    }
}
```

Entité Retrait

```
@Entity
@DiscriminatorValue("R")
public class Retrait extends Operation{
    public Retrait() { super();}
    public Retrait(Date dateOperation, double
    montant, Compte compte) {
        super(dateOperation, montant, compte);
    }
}
```

Interfaces DAO basées sur Spring Data

```
import org.springframework.data.jpa.repository.JpaRepository;
import ma.emsi.entities.Client;
public interface ClientRepository extends JpaRepository<Client, Long> {

}
```

```
import org.springframework.data.jpa.repository.JpaRepository;
import ma.emsi.entities.Compte;
public interface CompteRepository extends JpaRepository<Compte, String>{

}
```

```
import org.springframework.data.jpa.repository.JpaRepository;
import ma.emsi.entities.Operation;
public interface OperationRepository extends JpaRepository<Operation, Long>{

}
```

Test de la couche DAO

@SpringBootApplication

```
public class TpJpaApplication {
```

```
public static void main(String[] args) {
```

```
    ApplicationContext ctx= SpringApplication.run(TpJpaApplication.class, args);
```

```
    ClientRepository clientRepository= ctx.getBean(ClientRepository.class);
```

```
    /* * Ajouter 3 clients */
```

```
    clientRepository.save(new Client("Hassan")); clientRepository.save(new Client("AAAA"));
```

```
    clientRepository.save(new Client("CCCCC"));
```

code	nom
1	Hassan
2	AAAA
3	CCCCC

type_cpte	code	date_creation	solde	decouvert	taux	code_cli
CC	c1	2016-11-25 15:44:09	9000	800	NULL	1
CE	c2	2016-11-25 15:44:09	7800	NULL	5	1

```
    ComptRepository comptRepository= ctx.getBean(ComptRepository.class);
```

```
    /* * consulter un client sachant sont code */
```

```
    Client c1=clientRepository.findOne(1L);
```

```
    /* * Créer 2 comptes : un courant et l'autre épargne */
```

```
    comptRepository.save( new ComptCourant("c1", 9000, new Date(), c1,800.0));
```

```
    comptRepository.save( new ComptEpargne("c2", 7800, new Date(), c1,5.0));
```

```
    OperationRepository operationRepository=ctx.getBean(OperationRepository.class);
```

```
    /* * Consulter un compte sachant sont code */
```

```
    Compt cp=comptRepository.findOne("c1");
```

```
    /* * Effectuer des versments et des retraits sur le compte c1 */
```

```
    operationRepository.save(new Versement(new Date(),6000,cp));
```

```
    operationRepository.save(new Versement(new Date(),7000,cp));
```

```
    operationRepository.save(new Retrait(new Date(),5400,cp));
```

```
}}
```

type	numero	date_operation	montant	code_cpte
V	1	2016-11-25 15:44:09	6000	c1
V	2	2016-11-25 15:44:09	7000	c1
R	3	2016-11-25 15:44:09	5400	c1



Autres exemples d'associations



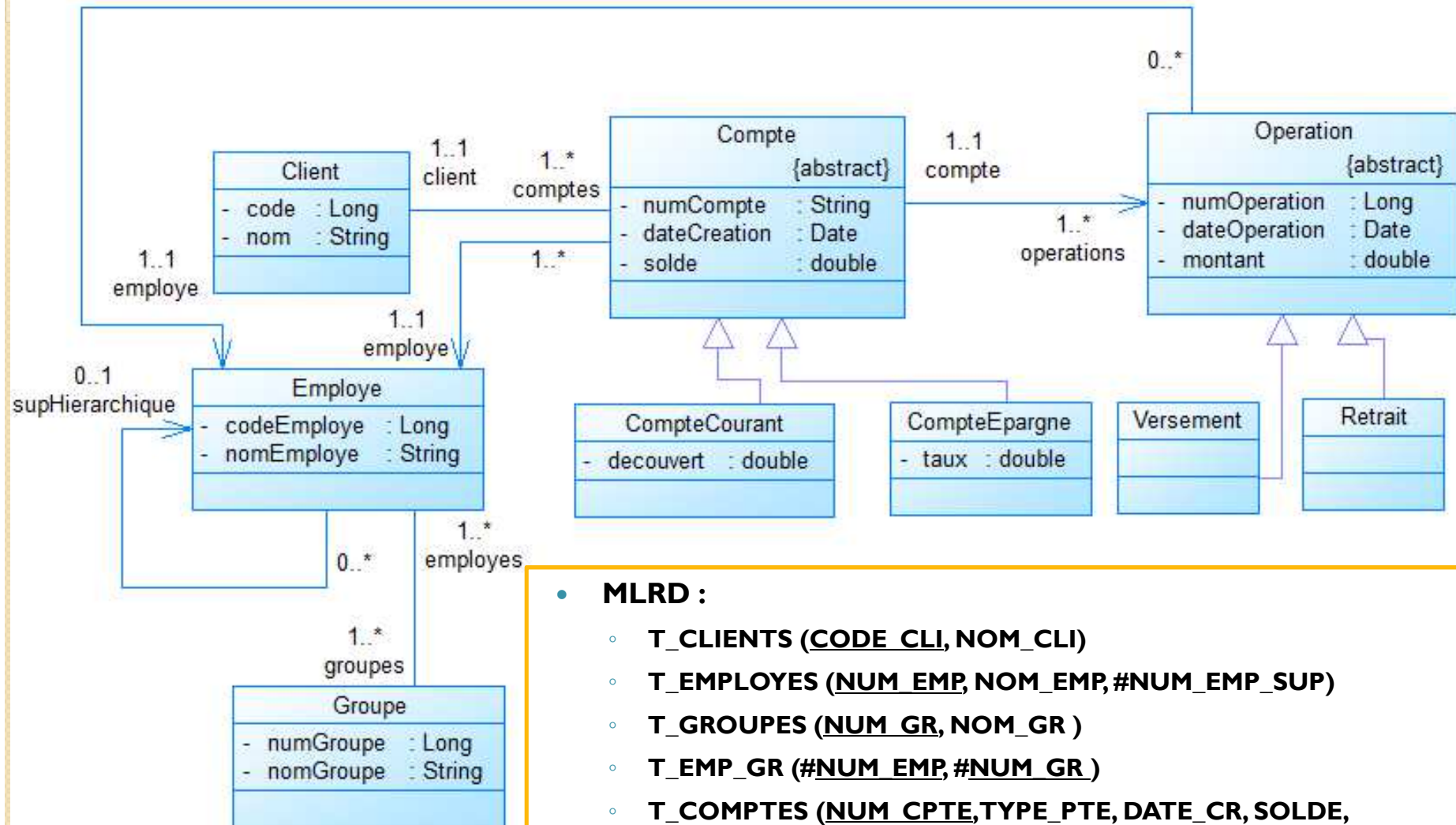
Gérer les associations et l'héritage entre les entités

- 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

Exemple de problème

- On souhaite créer une application qui permet de gérer des comptes bancaire.
 - Chaque compte est défini un numéro, un solde et une date de création
 - Un compte courant est un compte qui possède en plus un découvert
 - Un compte épargne est un compte qui possède en plus un taux d'intérêt.
 - Chaque compte appartient à un client et créé par un employé.
 - Chaque client est défini par son code et son nom
 - Un employé est défini par son code et sont solde.
 - Chaque employé possède un supérieur hiérarchique.
 - Chaque employé peut appartenir à plusieurs groupes
 - Chaque groupe, défini par un code est un nom, peut contenir plusieurs employés.
 - Chaque compte peut subir plusieurs opérations.
 - Il existe deux types d'opérations :Versement et Retrait
 - Chaque opération est effectuée par un employé.
 - Une opération est définie par un numéro, une date et un montant.

Diagramme de classes et MLDR



- **MLRD :**

- **T_CLIENTS (CODE_CLI, NOM_CLI)**
- **T_EMPLOYES (NUM_EMP, NOM_EMP, #NUM_EMP_SUP)**
- **T_GROUPES (NUM_GR, NOM_GR)**
- **T_EMP_GR (#NUM_EMP, #NUM_GR)**
- **T_COMPTES (NUM_CPTE, TYPE_PTE, DATE_CR, SOLDE, #NUM_EMP, #CODE_CLI)**
- **T_OPERATIONS (NUM_OP, TYPE_OP, DATE_OP, MONTANT, #NUM_EMP, #NUM_CPTE)**

Entity Client

```
package banque.metier;
import java.io.Serializable; import java.util.Collection;
import javax.persistence.*;
@Entity
@Table(name="CLIENTS")
public class Client implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name="CODE_CLI")
    private Long codeClient;
    @Column(name="NOM_CLI")
    private String nomClient;
    @OneToMany(mappedBy="client", fetch=FetchType.LAZY)
    private Collection<Compte> comptes;
    // Getters et Setters
    // Constructeur sans param et avec params
}
```

Entity Employe

```
package banque.metier;
import java.io.Serializable; import java.util.Collection;
import javax.persistence.*;
@Entity
public class Employe implements Serializable{
    @Id
    @GeneratedValue
    private Long numEmploye;
    private String nomEmploye;
    private double salaire;
    @ManyToOne
    @JoinColumn(name="NUM_EMP_SUP")
    private Employe supHierarchique;
    @ManyToMany
    @JoinTable(name="EMP_GROUPES", joinColumns =
    @JoinColumn(name = "NUM_EMP"),
    inverseJoinColumns = @JoinColumn(name = "NUM_GROUPE"))
    private Collection<Groupe> groupes;
    // Getters et Setters
    // Constructeur sans param et avec params
}
```

Entity Groupe

```
package banque.metier;

import java.io.Serializable; import
    java.util.Collection;
import javax.persistence.*;

@Entity
public class Groupe implements Serializable {
    @Id
    @GeneratedValue
    private Long numGroupe;
    private String nomGroupe;
    @ManyToMany(mappedBy="groupes")
    private Collection<Employe> employes;
    // Getters et Setters
    // Constructeur sans param et avec params
}
```

Entity Compte

```
package banque.metier;
import java.io.Serializable; import java.util.Collection;
import javax.persistence.*;
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="TYPE_CPTE",discriminatorType=DiscriminatorType.STRING,length=2)
public abstract class Compte implements Serializable {
    @Id
    private String numCompte;
    private Date dateCreation;
    private double solde;
    @ManyToOne
    @JoinColumn(name="CODE_CLI")
    private Client client;
    @ManyToOne
    @JoinColumn(name="NUM_EMP")
    private Employe employe;
    @OneToMany(mappedBy="compte")
    private Collection<Operation> operations;
    // Getters et Setters
    // Constructeur sans param et avec params
}
```

Entity CompteCourant

```
package banque.metier;  
import java.io.Serializable;  
import java.util.Collection;  
import javax.persistence.*;  
@Entity  
@DiscriminatorValue("CC")  
public class CompteCourant extends Compte{  
    private double decouvert;  
    // Getters et Setters  
    // Constructeur sans param et avec params  
}
```


Entity CompteEpargne

```
package banque.metier;  
import java.io.Serializable;  
import java.util.Collection;  
import javax.persistence.*;  
@Entity  
@DiscriminatorValue("CE")  
public class CompteEpargne extends Compte {  
    private double taux;  
    // Getters et Setters  
    // Constructeur sans param et avec params  
}
```

Entity Operation

```
package banque.metier;
import java.io.Serializable;
import java.util.Collection;
import javax.persistence.*;
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="TYPE_OP",discriminatorType=DiscriminatorType.STRING,length=2)
public abstract class Operation implements Serializable {
    @Id
    @GeneratedValue
    private Long numOperation;
    private Date dateOperation;
    private double montant;
    @ManyToOne
    @JoinColumn(name="NUM_CPTE")
    private Compte compte;
    @ManyToOne
    @JoinColumn(name="NUM_EMP")
    private Employe employe;
    // Getters et Setters
    // Constructeur sans param et avec params
}
```

Entity Versement

```
package banque.metier;  
import java.io.Serializable;  
import java.util.Collection;  
import javax.persistence.*;  
@Entity  
@DiscriminatorValue("V")  
public class Versement extends Operation{  
    // Constructeur sans param et avec params  
}
```

Entity Retrait

```
package banque.metier;  
import java.io.Serializable;  
import java.util.Collection;  
import javax.persistence.*;  
@Entity  
@DiscriminatorValue("R")  
public class Retrait extends Operation {  
    // Constructeur sans param et avec params  
}
```