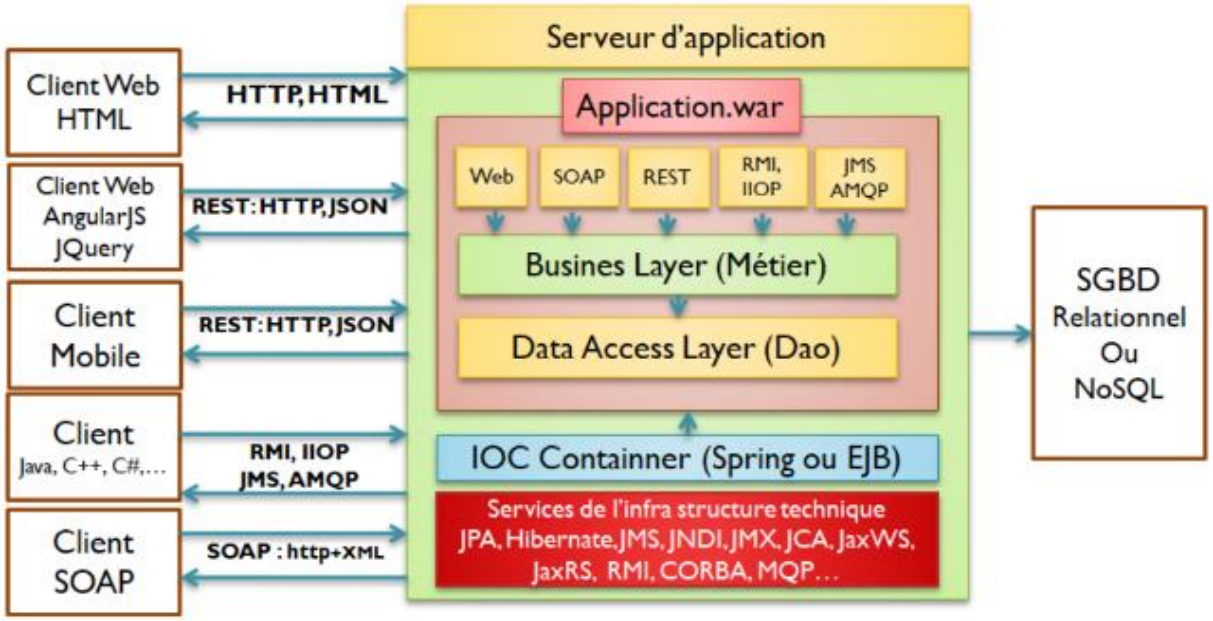
JEE: ORM, JPA, Hibernate, Spring Data

Architecture JEE :

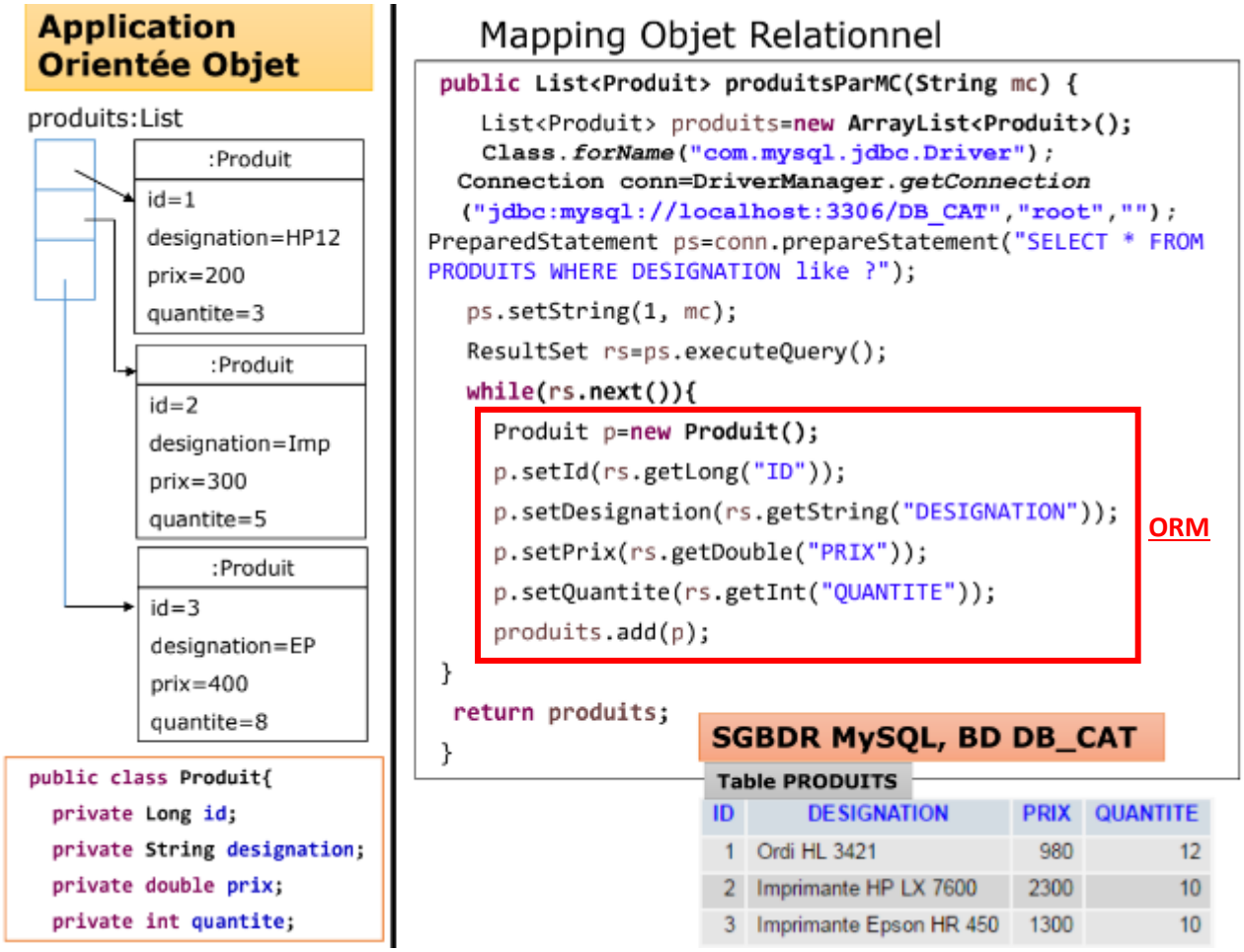
- **Persistant :** Un objet de nature est volatile(au run de l'app, l'objet est dans la RAM dès l'app s'arrête les données de l'objet seront perdus), un objet persistant c'est un objet dans l'état ses données sont enregistré quelque par soit dans un fichier, BD …, c-à-d même si l'app s'arrête en perdre pas les données de cette objet persistant.
- **SGBD :** Système de base de données relationnelles c-à-d les données sont stockées dans des tables et les tables sont liés par des relations à travers le concept clé primaire et clé étrangère.
- **ORM :** Mapping Objet Relationnel consiste à faire la correspondance entre les données stockées dans les SGBD(dans les tables) avec les objets de l'pp
- **JPA :** Java Persistence API
- **Hibernate :** Premier Framework crée pour ORM.
- **Spring Data :**
- **La couche Dao va se charger de faire le ORM**



➢ **Faire le mapping Objet relationnel sans JPA, Hibernate ou Spring Data**

- Utiliser le Driver(pilote, API) JDBC**(Java BD Connectivity)** pour connecter à la base de données MySql
- Faire la correspondance entre les données d'une requête SQL avec un Objet Produit attribut par attribut.
- Code se répète pour chaque classe

- Multi connexion à la base de données pour chaque classe créée.



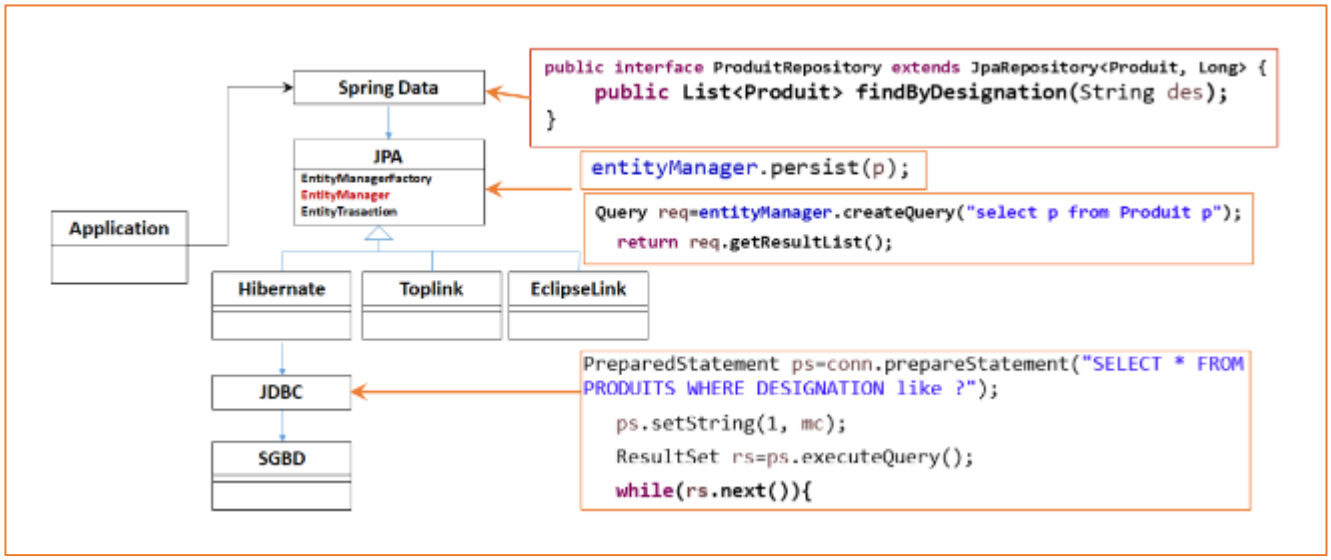
➢ **Alors**

Pourquoi il faut utiliser des Framework ORM

- **Hibernate** permet de gainer de temps car son but est de libérer le dev de 95% des tâches de programmation liées à a persistance des données communes.
- **Hibernate** assure la portabilité de l'app si on change de SGBD
- **Hibernate** assure la performance de l'app car il propose au dev des méthodes d'accès aux BD plus efficace.

Autre Framework qui fait le ORM : Toplink, EclipseLink, IBatis

- Tous les Framework de ORM respectent la même **spécification JPA**
- **JPA** est une **spécification** créée par Sun pour standardiser le ORM
- **JPA** est une **API** définit un ensemble d'interfaces, de classe abstraites et d'annotations qui permettent la description du ORM.
- **JPA est une API alors que Hibernate une classe qui implémente cette API JPA : Couplage faible**
  - > L'utilisation de JPA permet à l'app d'être indépendante du Framework ORM utilisé.
  - > L'implémentation de référence de JPA c'est EclipseLink créer par Sun pour test l'API JPA.



➢ **Faire le mapping Objet relationnel des entités avec JPA**

- Il existe deux moyens pour mapper les entités :
  - Créer des fichiers XML de mapping : a l'avantage de séparer le code java du ORM.
  - Utiliser les Annotations JPA : laisse le code indépendant de Hibernate.
- Quelque annotations JPA de Mapping des Entités

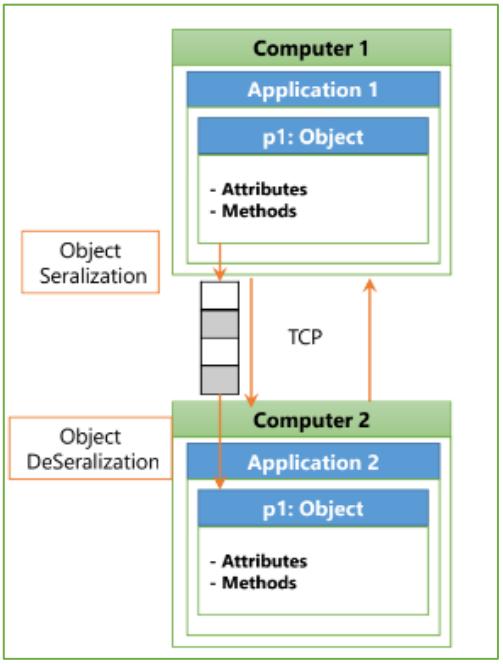
- @Entity : Indique que la classe est persistante est correspond à une table dans la base de donée
- @Table : Préciser le nom de la table concernée par le mapping. Par défaut c'est le nom de la classe qui sera considérée
- @Column : Associer un champ de la colone à la propriété. Par défaut c'est le nom 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
- @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
- @JoinedColumn : Pour décrire une clé étrangère dans une table
- @ManyToMany : Pour décrire une association plusieurs à plusieurs
- Etc...

- **Entité JPA / Java Bean :** est une classe souvent sérialisable avec des attributs privés, des setters, des getters et un constructeur par défaut(sans paramètres) et qui utilise deux annotation obligatoire **@Entity** et **@Id.**
- **La Sérialisation :** prendre l'objet existé dans la RAM le convertir à un tableau d'octets et le stocké dans un fichier.
- **La Désérialisation :** reconstruire l'objet envoyé octet par octet.
- **Implémente Serializable :** obligatoire dans le cas de micro-services(couche DAO dans M1, Couche métier dans M2).

Exemple :

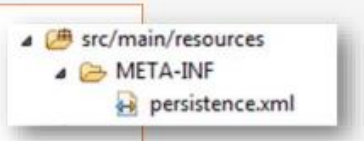
```java
package dao;
import java.io.Serializable; import javax.persistence.*;
@Entity
@Table(name="PRODUITS")
public class Produit implements Serializable {
  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  @Column(name="REF")
  private Long reference;
  @Column(name="DES")
  private String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

 // Constructeur par défaut
 // Getters et Setters
}
```



Configuration de l'unité de persistance : persistance.xml

```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.dialect" value="org.hibernate.dialect.MySQLDialect"/>
     <property name="hibernate.hbm2ddl.auto" value="update"/>
     <property name="hibernate.show_sql" value="true"/>
</properties>
  </persistence-unit>
</persistence>
```



- **dialect** : classe permet qui permet au Framework de savoir la syntaxe SQL qu'il faut utiliser pour ce SGBD
- **Dans le Langage SQL il y a 3 catégories :**
  - **DDL** : Data Definition Language (create, alter, delete table) pour créer la structure de BD
  - **DML** : Data Manipulation Language où on trouve les instruction insert, update, delete…
  - **DCL** : Data Control Language pour créer les user, les droits d'accès ….
- **hbm2ddl :**
  - **value = « update »** : au démarrage de l'app si une table n'est pas créée il va la créer si déjà créée il ne va pas la touché.

- **value = « create »** : si la table n'existe pas il va la créée si elle existe déjà il va la supprimer(écraser) et la recréer à nouveau.

Premier Objet créé au démarrage de l'app : EntityManagerFactory

- L'objet va lire le fichier de persistance « persistance.xml »
- Etablir une connexion avec la BF et génère les tables relatives aux entités.

EntityManagerFactory

```
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");
}}
```

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	

- **La diff entre HQL et SQL :** dans SQL on manipule les tables et les relations entre les tables alors que dans HQL on connu pas les tables mais on manipule les classes et les relation entre les classes.

JPA dans un projet Spring

- Spring est un Framework qui assure l'inversion de contrôle.
- Spring peut s'occuper du code technique comme la configuration de JPA et la gestion des transactions.
- Spring Boot est une version de Spring qui permet de simplifier à l'extrême :
  - La gestion des dépendances Maven : Spring Boot Starter
  - Auto Configuration : application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/db_cat_mvc?serverTimezone=UTC
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
```

Spring Data(Fait appel à JPA): à créer une interface générique **JpaRepository<T, V>** qui offre toutes les fonctions de manipulation de données(find, save …).

```
public interface ProduitRepository extends JpaRepository<Produit, Long> {
    public List<Produit> findByDesignation(String des);
}
```