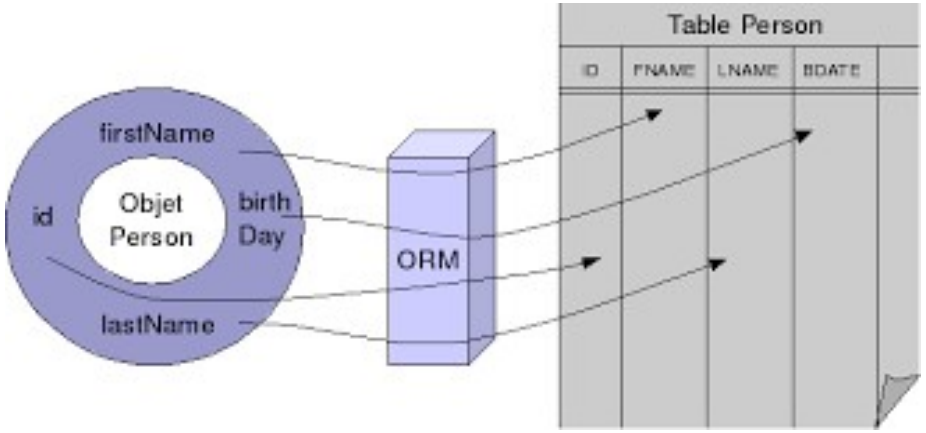
Java Avancé

Hibernate Framework (Object Relational Mapping)

A, Larhlimi

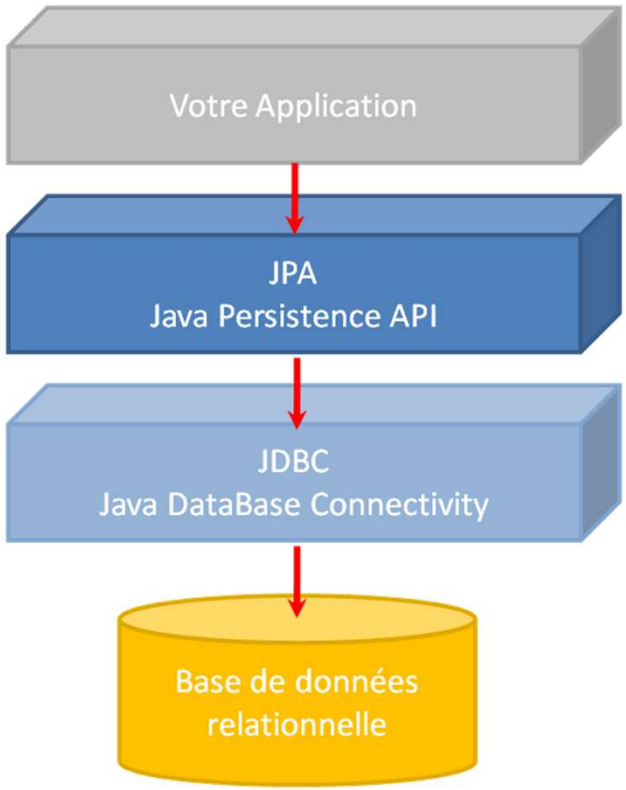
Introduction à Hibernate

- Hibernate est un framework ORM (Object Relational Mapping) pour Java.
- Il simplifie la gestion de la persistance des objets Java dans une base de données relationnelle.
- Permet d'éviter le code SQL manuel et de se concentrer sur la logique métier.

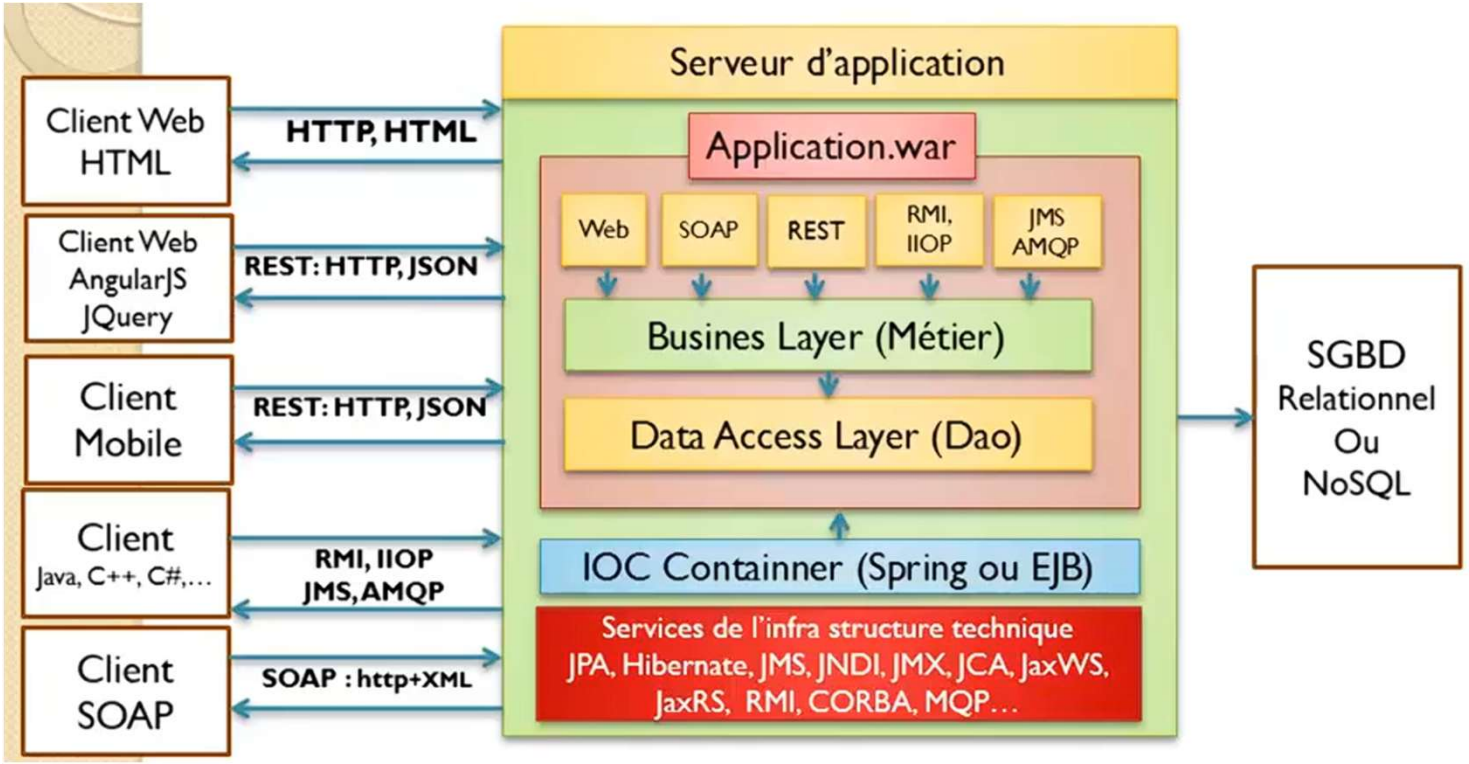


La persistance des données

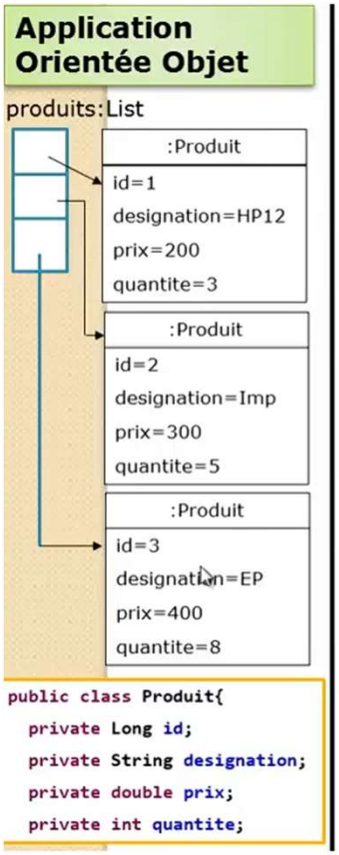
•**Abstraction quasi totale du langage SQL** : c'est le rôle de l'ORM de produire le code SQL équivalent à vos actions.
•**Indépendance vis-à-vis de la base de données utilisée** : comme vous ne produisez pas le code SQL, vous n'êtes pas lié à une base de données précise. Les ORM JPA peuvent travailler sur toutes les bases de données manipulables par JDBC. Vous êtes donc libre de changer votre SGBDr (Système de Gestion de Base de Données relationnel) à tout moment.
•**Meilleure productivité** : comparer à JDBC, vous avez beaucoup moins de code à produire (et notamment avec le SQL).
Risque d'injection SQL fortement réduit : vu que vous ne produisez plus de codes SQL, vous évitez un grand nombre d'attaques basées sur une mauvaise utilisation de ce langage.



Architecture Global



DAO




```
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

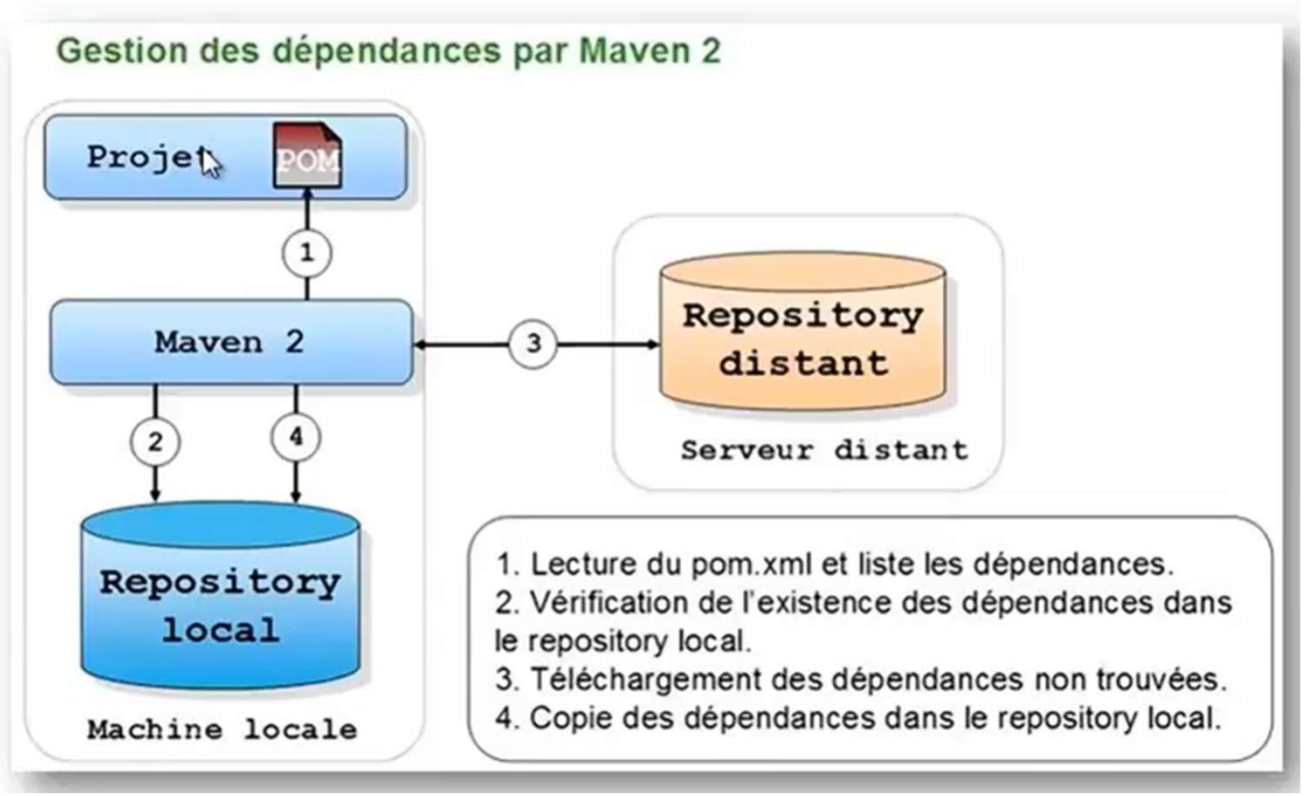
ID	DESIGNATION	PRIX	QUANTITE
1	Ordi HL 3421	980	12
2	Imprimante HP LX 7600	2300	10
3	Imprimante Epson HR 450	1300	10

Mapping +++++

```
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 connaitre la structure de la base de données, mais plutôt on connait 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

Principe de la Gestion des dépendances (Rappel Maven)



Objectifs de la présentation

- Comprendre le rôle et les avantages de Hibernate.
- Apprendre à configurer Hibernate dans un projet Java.
- Découvrir les principales opérations CRUD.
- Réaliser un TP guidé pour mettre en pratique les concepts.

Pourquoi utiliser Hibernate ?

- Réduction du code SQL.
- Portabilité entre SGBD.
- Gestion automatique des relations entre objets.
- Intégration facile avec Spring et JPA.

Architecture de Hibernate

- Les principaux composants : SessionFactory, Session, Transaction, Query.
- Hibernate utilise un cache pour optimiser les accès à la base de données.
- La configuration est centralisée dans un fichier XML ou via annotations.

Comparaison JDBC vs Hibernate

- JDBC : manipulation directe du SQL et des ResultSets.
- Hibernate : gestion des entités Java et du mapping automatique.
- Gain de productivité et maintenance facilitée avec Hibernate.

Installation et dépendances Maven

- Ajoutez la dépendance Hibernate dans le fichier pom.xml :
- <dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>6.4.1.Final</version>
  </dependency>
- Incluez également la dépendance du driver JDBC correspondant à votre SGBD.

Configuration du fichier hibernate.cfg.xml

- Exemple minimal :
- <hibernate-configuration>
  <session-factory>
  <property name='hibernate.connection.driver_class'>com.mysql.cj.jdbc.Driver</property>
  <property
  name='hibernate.connection.url'>jdbc:mysql://localhost:3306/testdb</property>
  <property name='hibernate.connection.username'>root</property>
  <property name='hibernate.connection.password'>root</property>
  <property name='hibernate.dialect'>org.hibernate.dialect.MySQLDialect</property>
  <mapping class='com.example.model.Etudiant'/>
  </session-factory>
  </hibernate-configuration>

Création de l'entité Java

- Exemple d'entité :
- ```
  @Entity
  @Table(name='etudiant')
  public class Etudiant {
   @Id
   @GeneratedValue(strategy=GenerationType.IDENTITY)
   private Long id;
   private String nom;
   private String prenom;
  }
  ```
- Chaque classe correspond à une table et chaque champ à une colonne.
  ```

Configuration via annotations

- Alternative à XML :
- Utiliser @Entity, @Table, @Column, @Id, @GeneratedValue, etc.
- Les annotations simplifient la maintenance et favorisent la cohérence du code.

SessionFactory et Session

- SessionFactory : créée une seule fois, gère les connexions.
- Session : représente une unité de travail, utilisée pour les opérations CRUD.
- Transaction : garantit la cohérence des modifications.

Opérations CRUD avec Hibernate

- Create : session.save(objet);
- Read : session.get(Classe.class, id);
- Update : session.update(objet);
- Delete : session.delete(objet);
- Toujours encadrer par une transaction.

Langage HQL (Hibernate Query Language)

- Syntaxe proche du SQL, mais orientée objet.
- Exemple : from Etudiant where nom = 'Dupont'
- Permet des jointures, agrégations et sous-requêtes.

Gestion des relations entre entités

- @OneToMany, @ManyToOne, @OneToOne, @ManyToMany
- Hibernate gère les clés étrangères et jointures automatiquement.
- Possibilité de cascade (ex: cascade=ALL).

Cycle de vie d’une entité

- États : transient, persistent, detached, removed.
- Les objets passent d’un état à l’autre selon les opérations réalisées.
- Hibernate gère la synchronisation entre objet et base de données.

TP : Contexte du projet

- Création d'une application de gestion d'étudiants.
- Objectif : ajouter, consulter, modifier et supprimer des étudiants dans une base MySQL.
- Le TP se fera étape par étape.

TP Étape 1 : Configuration du projet

- Créer un projet Maven.
- Ajouter les dépendances Hibernate et MySQL.
- Configurer hibernate.cfg.xml et créer l’entité Etudiant.

TP Étape 2 : DAO et SessionFactory

- Créer une classe HibernateUtil pour initialiser la SessionFactory.
- Créer un DAO EtudiantDAO avec les méthodes CRUD.
- Tester les opérations dans une classe Main.

TP Étape 3 : Utilisation du HQL

- Créer une méthode pour lister tous les étudiants :
- List<Etudiant> etudiants = session.createQuery('from Etudiant').list();
- Afficher les résultats dans la console.

TP Étape 4 : Améliorations possibles

- Intégrer une interface graphique (JavaFX ou Swing).
- Ajouter la validation des données.
- Étendre le modèle avec des entités supplémentaires (Cours, Notes).

Conclusion

- Hibernate simplifie la gestion des données en Java.
- Réduit le couplage avec la base de données.
- Facilite la maintenance et l’évolution des projets.