

Framework de persistance

JPA: Java Persistence Api

A. MADANI

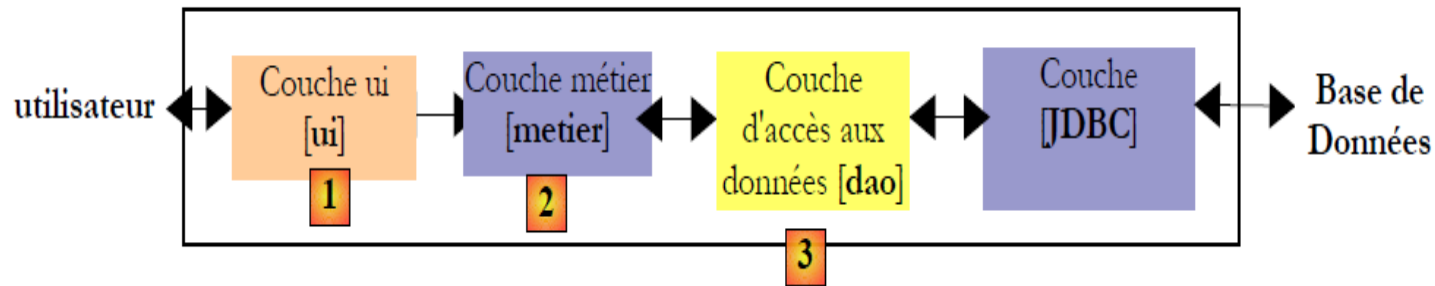
Madani.a@ucd.ac.ma

ORM JPA (1)

- Introduction
- Gestion de la correspondance objet/relationnel
- ORM: Object Relationnel Mapping
- Framework: Hibernate, Tooplik,...
- JPA: Java persistance API
- Mapping Objet Relationnel Avec JPA
 - Les entités
 - Annotations
 - Fichier de configuration

Introduction

- Architecture multicouches



- La couche [1] contient les interfaces utilisateur (UI : User Interfaces)
- La couche [2] est la partie métier : Classes métiers
- La couche [3], appelée [dao] (Data Access object) est la couche qui fournit à la couche [2] des données pré-enregistrées (fichiers, bases de données, ...) et qui enregistre certains des résultats fournis par la couche [2].

Introduction

- La couche DAO gère des données provenant de deux mondes:
 - Monde relationnel (BDR)
 - Monde Objet (Couche métier)
- Comment gérer la persistance des objets provenant de la couche métier?
- Il faut faire la correspondance entre les deux mondes.
- Solutions:
 - Manuelle (solution 1 : c'est ce que nous avons fait jusqu'ici)
 - Utilisation d'un framework de mapping O/R (solution 2)

Gestion de la correspondance objet/relationnel (solution 1)

- 30% du code java est consacré à gérer la mise en rapport SQL/JDBC et la non correspondance entre le modèle objet et le modèle relationnel
 - Gestion des associations
 - Correspondances de types

- Par exemple :

SQL/JDBC:

```
String req= "insert into personne values (" + id + ", ' " + nom + " ', ' " + prenom + " ' " + " );";
```

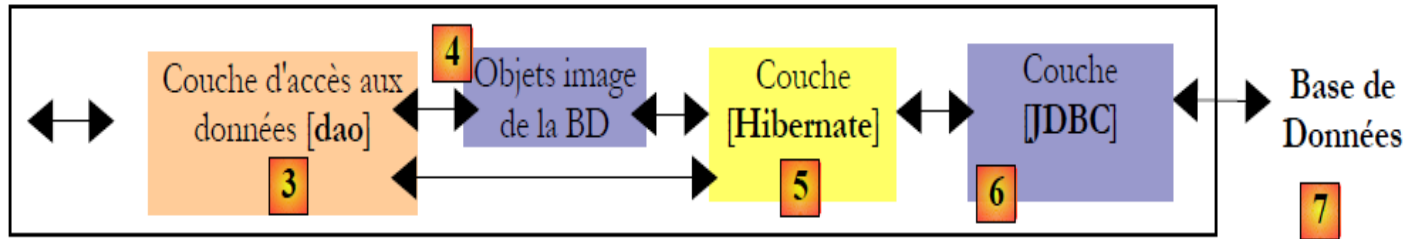
Gestion de la correspondance objet/relationnel (solution 2)

- Le mapping Objet/Relationnel (ORM) désigne la persistance automatisée et transparente d'objets dans une application Java vers les tables d'une base de données relationnelle à l'aide des méta données décrivant le mapping entre les objets et la base de données.
- Les outils et frameworks ORM réduisent grandement le code qui permet cette correspondance et facilitent donc les adaptations mutuelles des deux modèles.
- Exemple de frameworks:
 - Hibernate
 - Toplink,...

ORM : Object Relationnel Mapping

- C'est une technique de mapping d'une représentation des données d'un modèle objet vers un modèle relationnel de base de données et inversement.
- Quatre éléments constituant une solution ORM :
 1. Une API pour effectuer les opérations **CRUD** (**Create, Read, Update, delete**) de base sur les objets des classes persistantes.
 2. Un langage ou une API pour spécifier des requêtes qui se réfèrent aux classes et aux propriétés des classes.
 3. Un système pour spécifier des **métadonnées de mapping**.
 4. Une technique pour l'implémentation du mapping objet/relationnel permettant d'interagir avec des objets

Solutions ORM: Hibernate



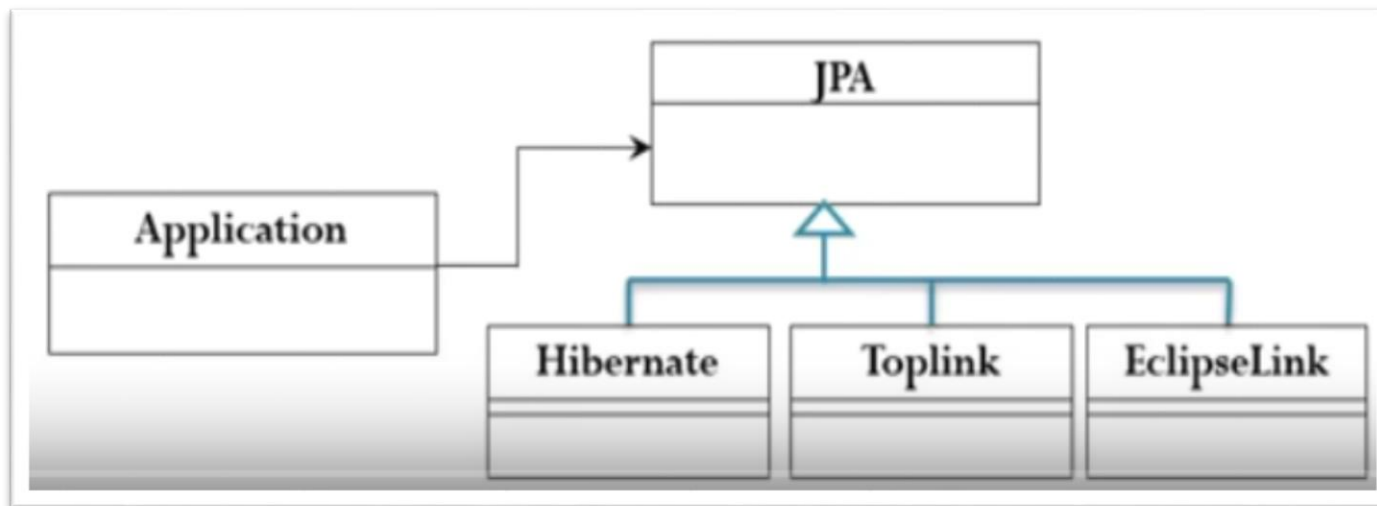
- Hibernate est un ORM (Object Relational Mapping) qui fait le pont entre le monde relationnel des bases de données et celui des objets manipulés par Java.
- Le développeur de la couche [dao] ne voit plus la couche [Jdbc] ni les tables de la base de données.
 - Il ne voit que l'image objet de la base de données, image objet fournie par la couche [Hibernate].
- XML constitue le format de description de la correspondance entre les tables relationnelles et les classes Java.

Hibernate (suite)

- Hibernate fournit au développeur, un langage HQL (Hibernate Query Language) pour interroger le contexte de persistance [4] et non la BD elle-même.
- Limites:
- Hibernate est populaire mais complexe à maîtriser:
 - Dès qu'on a une base de données avec des tables ayant des relations un-à-plusieurs ou plusieurs-à-plusieurs, la configuration du pont relationnel/objets devient complexe.
 - Des erreurs de configuration peuvent alors conduire à des applications peu performantes.
- Apparition des plusieurs ORMs
 - Toplink
 - JDO
 - Eclipse Link
 - Ibatis
 - etc

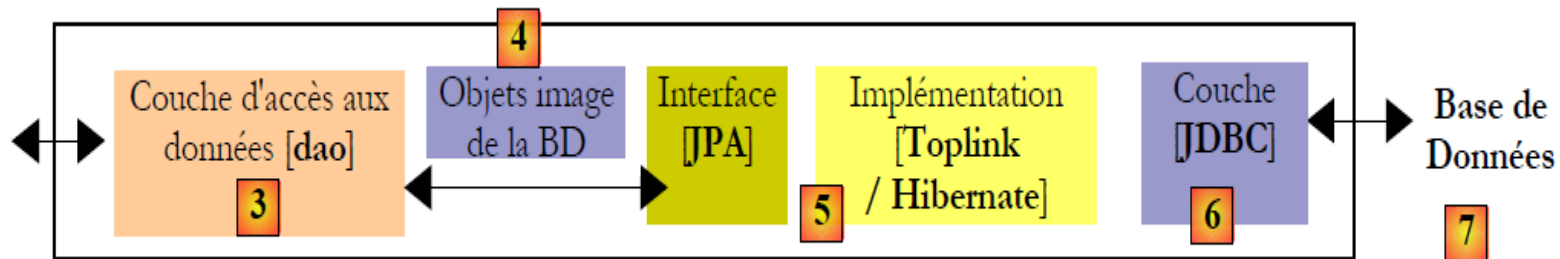
JPA: Java persistence API (1)

- Devant le succès des produits ORM, Sun le créateur de Java, a décidé de standardiser une couche ORM via une spécification appelée JPA apparue en même temps que Java 5:



JPA: Java persistence API (1)

- Avec JPA, l'architecture précédente devient la suivante :




- **Avantages:**
- Avant, la couche [dao] est très dépendante du Framework ORM choisi.
 - Si le développeur décide de changer sa couche ORM, il doit également changer sa couche [dao] qui avait été écrite pour dialoguer avec un ORM spécifique
- Maintenant, il va écrire une couche [dao] qui va dialoguer avec une couche JPA.
 - Quelque soit le produit qui implémente celle-ci, l'interface de la couche JPA présentée à la couche [dao] reste la même

Mapping Objet Relationnel Avec JPA

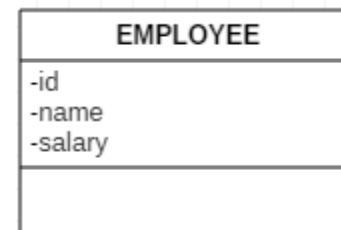
- La communication entre les mondes objet et relationnel suppose une transformation pour adapter la structure des données relationnelles au modèle objet.
- Les Entités
 - Les classes dont les instances peuvent être persistantes sont appelées des entités dans la spécification de JPA
 - Le développeur indique qu'une classe est une entité en lui associant l'annotation **@Entity**

Les entités: Exemple1

- Considérons une base de données ayant une unique table [**EMP**] dont le rôle est de mémoriser quelques informations sur des employés que nous voulons mapper avec la classe [**EMPLOYEE**] :
- La table **EMP** contient les champs suivants:
 - **ID**: clé primaire de la table
 - **EMP_NAME** : nom de la personne
 - **SALARY** : son salaire

Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
ID 	int(11)			Non	Aucun(e)		AUTO_INCREMENT
EMP_NAME	varchar(20)	utf8mb4_general_ci		Non	Aucun(e)		
SALARY	float			Non	Aucun(e)		

- La classe EMPLOYEE est caractérisée par les attributs suivants :
 - **id** : Identifiant de l'employé
 - **name** : nom de l'employé
 - **salary** : son salaire



Les entités: Exemple1 (suite)

- L'objet [EMPLOYEE] image de la table [EMP] présentée précédemment pourrait être le suivant :

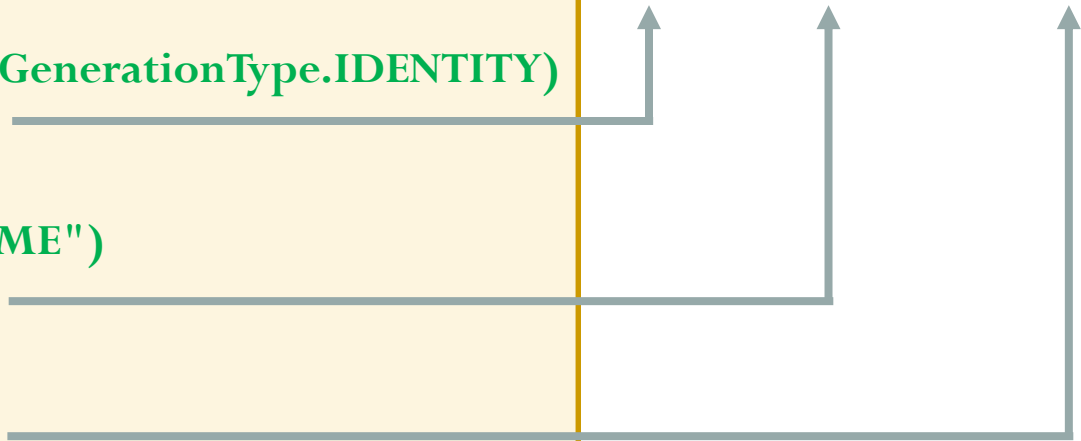
```
@Entity
@Table(name = "EMP")
public class EMPLOYEE {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Column(name = "EMP_NAME")
    private String name;
    private double salary;
    public EMPLOYEE() {
        super();
    }
    public int getId() {return id;}
    public void setId(int id) {this.id = id;}
    public String getName() {return name;}
    public void setName(String name) {this.name = name;}
    public double getSalary() {return salary;}
    public void setSalary(double salary) {this.salary = salary;}
}
```

Java Persistence API: *Mappings de base*

```
@Entity  
@Table(name = "EMP")  
public class EMPLOYEE {  
  @Id  
  @GeneratedValue(strategy = GenerationType.IDENTITY)  
  private int id;  
  
  @Column(name = "EMP_NAME")  
  private String name;  
  
  private double salary;  
  
  ....  
}
```

EMP

ID	EMP_NAME	SALARY



Annotation JPA

- L'annotation **@Entity** est la première annotation indispensable. Elle se place avant la ligne qui déclare la classe et indique que la classe en question doit être gérée par la couche de persistance JPA.
 - En l'absence de cette annotation, toutes les autres annotations JPA seraient ignorées.
- L'annotation **@Table** désigne la table de la base de données dont la classe est une représentation. Son principal argument est `name` qui désigne le nom de la table.
 - En l'absence de cet argument, la table portera le nom de la classe
- L'annotation **@Id** sert à désigner le champ dans la classe qui est image de la clé primaire de la table. Cette annotation est obligatoire.

Annotation JPA (suite)

- L'annotation **@Column** sert à faire le lien entre un attribut de la classe et la colonne de la table dont le champ est l'image.
 - L'attribut **name** indique le nom de la colonne dans la table. En l'absence de cet attribut, la colonne porte le même nom que le champ.
 - L'argument **nullable=false** indique que la colonne associée au champ ne peut avoir la valeur NULL et que donc le champ doit avoir nécessairement une valeur.
- L'annotation **@GeneratedValue** indique comment est générée la clé primaire lorsqu'elle est générée automatiquement par le SGBD.

Fichier de configuration

- **Un seul fichier de configuration XML**
- **IL est nécessaire d'indiquer au fournisseur de persistance comment il peut se connecter à la base de données**
- **Ce fichier peut aussi comporter d'autres informations**

Fichier de configuration (Suite)

```
<persistence-unit name="Application1">
  <class>DAO.EMPLOYEE</class>
  <properties>
    <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/webjpa"/>
    <property name="javax.persistence.jdbc.user" value="root"/>
    <property name="javax.persistence.jdbc.password" value=""/>
    <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
    <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect"/>
    <property name="hibernate.hbm2ddl.auto" value="update"/>
    <property name="hibernate.show_sql" value="true"/>
  </properties>
</persistence-unit>
```