TP N°2

Hibernate

Préparée par : Abdelmoughith Elaoumari, Yahya Lemkharbech et Hiba Dadda

Objectif : Bienvenue dans ce Travaux Pratiques dédié à la gestion des relations de mapping et aux opérations d'insertion avec Hibernate et Java. Hibernate, un Framework de persistance pour Java, facilite l'interaction entre les applications Java et les bases de données relationnelles.

Rappel sur Hibernate :

Hibernate simplifie la persistance des objets Java en les liant à une base de données relationnelle. Il offre une couche d'abstraction entre l'application et la base de données, permettant un développement plus orienté objet.

Objectif du TP : Dans ce TP, nous nous concentrerons sur deux aspects cruciaux de Hibernate : la définition des relations de mapping entre les entités et l'exécution d'opérations d'insertion dans la base de données.

Relations de Mapping : Nous explorerons les types de relations les plus couramment utilisés :

1. One-to-One : Définition d'une relation un à un entre deux entités.

2. One-to-Many : Établissement d'une relation un à plusieurs entre deux entités.

3. Many-to-One : Configuration d'une relation plusieurs à un entre deux entités.

4. Many-to-Many : Gestion d'une relation plusieurs à plusieurs entre deux entités.

Insertions et Exécution :

Après avoir défini nos relations de mapping, nous mettrons en pratique nos connaissances en écrivant un programme Java. Ce programme utilisera Hibernate pour insérer des données dans la base de données, suivies de la vérification des relations nouvellement créées. L'exécution de ce programme nous permettra de voir Hibernate en action, simplifiant le processus d'interaction avec la base de données.

Conseils :

- Suivez attentivement les étapes pour chaque relation. - Profitez de cette opportunité pour explorer et expérimenter avec Hibernate. - N'hésitez pas à ajuster les exemples en fonction de vos besoins spécifiques.

Prêts à plonger dans le monde des relations de mapping et des opérations d'insertion avec Hibernate et Java ? Allons-y !

1 Problématique

Vous êtes chargé de développer une application de gestion des étudiants dans une université. Chaque étudiant a une adresse unique et est inscrit dans une seule classe. De plus, chaque étudiant peut être inscrit à plusieurs cours, et chaque cours peut être suivi par plusieurs étudiants.

Votre tâche consiste à concevoir et implémenter le mappage des relations entre les entités : étudiant, adresse, classe et cours en utilisant le Framework Hibernate. Vous devrez créer les classes Java correspondantes avec les annotations Hibernate appropriées pour réaliser le mappage des relations.

Représentation UML : Diagramme de classes

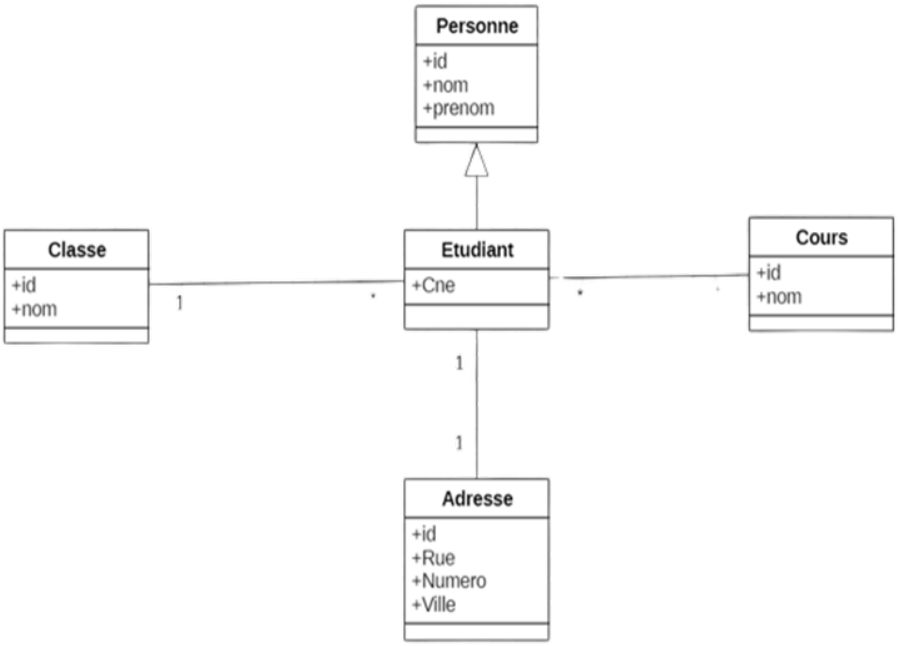

FIGURE 1 – Diagramme de classes

2 Relation d'héritage entre la classe 'Personne' et 'Etudiant'

Étape 1 : Ajout de l'annotation @Entity à la classe Personne et @Inheritance(strategy = InheritanceType.JOINED)

Dans la classe Personne, ajoutez l'annotation @Entity pour indiquer qu'il s'agit d'une entité persistante :

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Personne {
    // attributs et m thodes de la classe Personne
}

```

FIGURE 2 – Annotations à ajouter dans la classe personne

Étape 2 : Ajout de l'annotation @Entity et "extends" à la classe Etudiant

Dans la classe Etudiant, ajoutez l'annotation @Entity pour indiquer qu'il s'agit d'une entité persistante :

```java
@Entity
public class Etudiant extends Personne {
    // attributs et m thodes de la classe Etudiant
}
```

FIGURE 3 – Annotations à ajouter dans la classe étudiant

Remarques :

- L'annotation @Entity indique que la classe est une entité persistante.
- Avec ces annotations, Hibernate saura comment gérer la relation d'héritage entre les entités Personne et Etudiant.
-L'annotation "@Inheritance(strategy = InheritanceType.JOINED)" est utilisée en Java pour définir la stratégie d'héritage lors de la persistance d'objets dans une base de données relationnelle. La stratégie "JOINED" indique que chaque sous-classe est stockée dans une table distincte, avec une relation un-à-un entre les tables à l'aide de clés étrangères. Cette approche offre une meilleure normalisation de la base de données et permet de représenter des hiérarchies complexes d'objets.

Indication 1 : App.java (main)

N'oubliez pas de mettre à jour votre base de données en conséquence avec ces modifications. Après cela,

```java
public static void main( String[] args ) {
    Etudiant E = new Etudiant();
    E.setNom("Karima");
    E.setPrenom("karimi");
    E.setCne("E622222");

    Configuration con = new Configuration().configure()
    .addAnnotatedClass(Etudiant.class);
    SessionFactory sf = con.buildSessionFactory();
    Session ss = sf.openSession();
    Transaction tr = ss.beginTransaction();

    try {
        ss.save(E);
        tr.commit();
        System.out.println("Etudiant s'ins re avec succ s");
    } catch (Exception e) {
        System.out.println("Erreur lors de l'insertion : "
        + e.getMessage());
    } finally {
        ss.close();
    }
}

```

FIGURE 4 – Main pour l'insertion d'étudiant

vous pouvez procéder à l'écriture de votre application pour tester cette relation d'héritage et effectuer des opérations d'insertion.

3 Relation OneToOne entre la classe 'Etudiant' et la classe 'Adresse' :

Étape 1 : Ajout des Annotations @OneToOne dans la Classe Etudiant
ajoutez les annotations**@OneToOne** et **@JoinColumn** sur le champ adresse pour indiquer la relation et la clé étrangère :

```
 @OneToOne(cascade = CascadeType.ALL)
@JoinColumn(name = "adresse_id") // C l    trangre   vers l'adresse
private Adresse adresse;

```

FIGURE 5 – Les annotations à ajouter dans la classe d'étudiant

Étape 2 : Ajout de l'Annotation @OneToOne dans la Classe Adresse Dans la classe Adresse, ajoutez l'annotation @OneToOne sur le champ etudiant pour spécifier la relation inverse :

```
@OneToOne(mappedBy = "adresse")
private Etudiant etudiant;

```

FIGURE 6 – Les annotations à ajouter dans la classe addresse

Remarques :

- L'annotation **@OneToOne** avec **@JoinColumn** dans Etudiant indique que la relation est gérée côté Etudiant et qu'il y a une clé étrangère adresse-id dans la table etudiant qui référence la table adresse.

- L'annotation **@OneToOne** avec mappedBy dans Adresse spécifie que la relation est déjà gérée côté Etudiant par le champ adresse. Cela indique la propriété inverse de la relation. Avec ces annotations, Hibernate saura comment gérer la relation One-to-One entre les entités

Avec ces annotations, Hibernate saura comment gérer la relation One-to-One entre les entités Etudiant et Adresse.
Indication 2 : App.java (main)

```
1
2  public class App
3  {
4  public static void main( String[] args )
5  {
6  Address add =new Address();
7  add.setNumero(12);
8  add.setRue("Massira");
9  add.setVille("safi");
10 Etudiant E =new Etudiant();
11 E.setNom("Karima");
12 E.setPrenom("Karimi");
13 E.setCne("E622222");
14 add.setEtudiant(E);
15 E.setAddress(add);
16 Configuration con =new
17 Configuration().configure().addAnnotatedClass(Etudiant.class).addAnnotatedClass(
       Address.class);
18 SessionFactory sf =con.buildSessionFactory();
19 Session ss= sf.openSession();
20 Transaction tr=ss.beginTransaction();
21 }
22 try {
23 ss.save(E);
24 tr.commit();
25 System.out.println(E);
26 } catch (Exception e)
27 System.out.println("Erreur lors de l'insertion : " + e.getMessage());
28 } finally
29 ss.close();
30
31 }}
32
```

FIGURE 7 – Main de mapping entre addresse et etudiant

NB : N'oubliez pas de mettre à jour votre base de données en conséquence avec ces modifications. Après cela, vous pouvez procéder à l'écriture de votre application pour tester cette relation One-to-One et effectuer des opérations

4 Relation OneToMany et ManyToOne entre les classes 'Etudiant' et 'Classe'

Étape 1 : Ajout de l'Annotation @OneToMany dans la Classe Classe

Dans la classe Classe, ajoutez l'annotation @OneToMany sur le champ etudiants pour indiquer la relation OneToMany :

```
1
2    @OneToMany(mappedBy = "classe", cascade = CascadeType.ALL)
3  private List<Etudiant> etudiants;
4
```

FIGURE 8 – Les annotations à ajouter dans la classe Classe

Cette annotation indique que la classe Classe a une relation OneToMany avec la classe Etudiant, et que la propriété inverse de cette relation est définie par le champ classe dans la classe Etudiant.

Étape 2 : Ajout de l'Annotation @ManyToOne dans la Classe Etudiant
Dans la classe Etudiant, ajoutez l'annotation @ManyToOne sur le champ classe pour spécifier la relation ManyToOne :

```
@ManyToOne(cascade = CascadeType.ALL)
private Classe classe;

```

FIGURE 9 – Les annotations à ajouter dans la classe Etudiant

Cette annotation indique que la classe Etudiant a une relation ManyToOne avec la classe Classe, et que la clé étrangère associée est classe$_id$.
Remarques :

- L'annotation @OneToMany dans la classe Classe indique que cette classe a plusieurs étudiants associés à elle. La propriété mappedBy spécifie la propriété inverse de la relation, c'est-à-dire le champ classe dans la classe Etudiant.

- L'annotation @ManyToOne dans la classe Etudiant indique que chaque étudiant appartient à une seule classe. La clé étrangère est spécifiée avec @JoinColumn(name = "classe-id")
faisant référence à la colonne classe-id dans la table etudiant.
Indication 3 : App.java (main)

```java
public class App {
public static void main(String[] args) {
// Configuration de la session Hibernate
Configuration configuration = new
Configuration().configure().addAnnotatedClass(Etudiant.class)
.addAnnotatedClass(Classe.
SessionFactory sessionFactory = configuration.buildSessionFactory();
// Cr ation de la classe
Classe classe = new Classe();
classe.setNom("Informatique");
// Cr ation des  tudiants  et de leurs adresses
Etudiant etudiant1 = new Etudiant();
etudiant1.setNom("Ahmed");
etudiant1.setNom("karimi");
etudiant1.setCne("CNE001");
etudiant1.setClasse(classe);
Etudiant etudiant2 = new Etudiant();
etudiant2.setNom("Jasmine");
etudiant2.setNom("Jasmine");
etudiant2.setCne("CNE002");
etudiant2.setClasse(classe);
Etudiant etudiant3 = new Etudiant();
etudiant3.setNom("Ikram");
etudiant3.setNom("Ikrami");
etudiant3.setCne("CNE003");
etudiant3.setClasse(classe);

List<Etudiant> etudiants = new ArrayList<Etudiant>();
      etudiants.add(etudiant1);
      etudiants.add(etudiant2);
            etudiants.add(etudiant3);
      classe.setEtudiants(etudiants);

// Enregistrement des objets dans la base de donn es
Session session = sessionFactory.openSession();
Transaction transaction = session.beginTransaction();
try {
session.save(classe); // L'enregistrement de la classe entra nera  galement  l'
    enregistrement des  tudiants  et de leurs adresses
// Validation de la transaction
transaction.commit();
System.out.println("Les  tudiants  ont  t  ins r s avec succ s dans la classe
    !");
} catch (Exception e) {
// En cas d'erreur, annulation de la transaction
transaction.rollback();
System.out.println("Une erreur s'est produite lors de l'insertion des  tudiants  :
    " + e.getMessage());
} finally {
// Fermeture de la session Hibernate
session.close();
sessionFactory.close();
}
```

FIGURE 10 – Main de mapping OneToMany

5 Relation ManyToMany entre les classes 'Etudiant' et 'Cours'

Étape 1 : Ajout de l'Annotation @ManyToMany dans la Classe Etudiant Dans la classe Etudiant, ajoutez l'annotation @ManyToMany sur le champ cours pour indiquer la relation ManyToMany :

```
@JoinTable(
name = "etudiant_cours", // Nom de la table de jointure
joinColumns = @JoinColumn(name = "etudiant_id"), // Cl    trangre   vers la table
    etudiant
inverseJoinColumns = @JoinColumn(name = "cours_id") // Cl    trangre   vers la
    table cours
)
private List<Cours> cours;

```

FIGURE 11 – Les annotations à ajouter dans la classe Etudiant

Cette annotation indique que la classe Etudiant a une relation ManyToMany avec la classe Cours. La table de jointure etudiant-cours est spécifiée avec les colonnes etudiant.id et cours.id pour les clés étrangères.

Étape 2 : Ajout de l'Annotation @ManyToMany dans la Classe Cours

Dans la classe Cours, ajoutez l'annotation **@ManyToMany** sur le champ etudiants pour spécifier la relation Many To Many :

```
@ManyToMany
cascade CascadeType ALL mappedBy "
private List<Etudiant> etudiants;

```

FIGURE 12 – Les annotations à ajouter dans la classe cours

Remarques :

- L'annotation @ManyToMany dans la classe Etudiant indique que chaque étudiant peut être associé à plusieurs cours, et chaque cours peut avoir plusieurs étudiants. La table de jointure etudiant-cours gère cette relation.

-L'annotation @ManyToMany dans la classe Cours indique que chaque cours peut être associé à plusieurs étudiants, et chaque étudiant peut suivre plusieurs cours. La propriété mappedBy indique la propriété inverse gérée par la classe Etudiant. Avec ces annotations, Hibernate saura comment gérer la relation ManyToMany entre la classe Etudiant et la classe Cours.

Indication 4 : App.java (main)

```
public class App
{
public static void main( String[] args )
{
//Etudiant 1
Etudiant E1 =new Etudiant();
E1.setNom(" ahmed
E1.setPrenom(" karimi
E 1.setCne("K1562721");
//Etudiant 2
Etudiant E2 =new Etudiant();
E2.setNom(" ikram
E2.setPrenom(" ikrami
E2.setCne("21SS");
//la listes des etudiants
ArrayList<Etudiant> Etudiants = new ArrayList<Etudiant>();
Etudiants.add(E1);
Etudiants.add(E2);
// Cours1
Cours cr1 = new Cours();
cr1.setNom("java");
cr1.setEtudiants(Etudiants);
// Cours2
Cours cr2 = new Cours();
cr2.setNom("UML");
cr2.setEtudiants(Etudiants) ;
// listes des cours
ArrayList<Cours> courses = new ArrayList<Cours>();
courses.add(cr1);
courses.add(cr2);
E1.setCours(courses);
E2.setCours(courses);
Configuration con =new
Configuration().configure().addAnnotatedClass(Etudiant.class).addAnnotatedClass(
    Cours.class);
SessionFactory sf =con.buildSessionFactory();
Session ss= sf.openSession();
Transaction tr=ss.beginTransaction();
try {
for(Etudiant e :Etudiants) {
ss.save(e);
}
tr.commit();
System.out.println("Insertion avec succes");
} catch (Exception e) {
System.out.println("Erreur lors de l'insertion : " + e.getMessage());
} finally{
ss.close();
}
}}

```

FIGURE 13 – Les annotations à ajouter dans la classe Classe