One To Many JPA

Objectif

Comprendre la relation one to many

Utiliser les annotations :

```
@Entity
@Id
@GeneratedValue
@OneToMany
@ManyToOne
@JoinColumn
```

Ce que l'on veut obtenir

Associer un professeur à plusieurs étudiants

Vérifier que deux tables sont créées.

Vérifier que la récupération d'un étudiant récupère également professeur associé

Utiliser cascade

Constater la notion de lazy loading par défaut

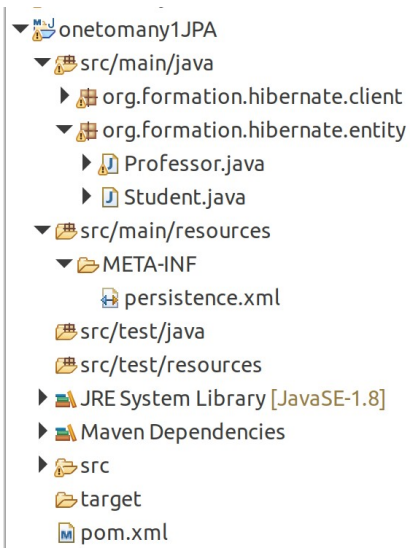
Comprendre la notion de inverse end et de owner

Permettre la mise à jour du côté de l'inverse end

Action

Configuration du projet

Créer un projet maven simple et le configurer pour JPA avec les packages suivants :

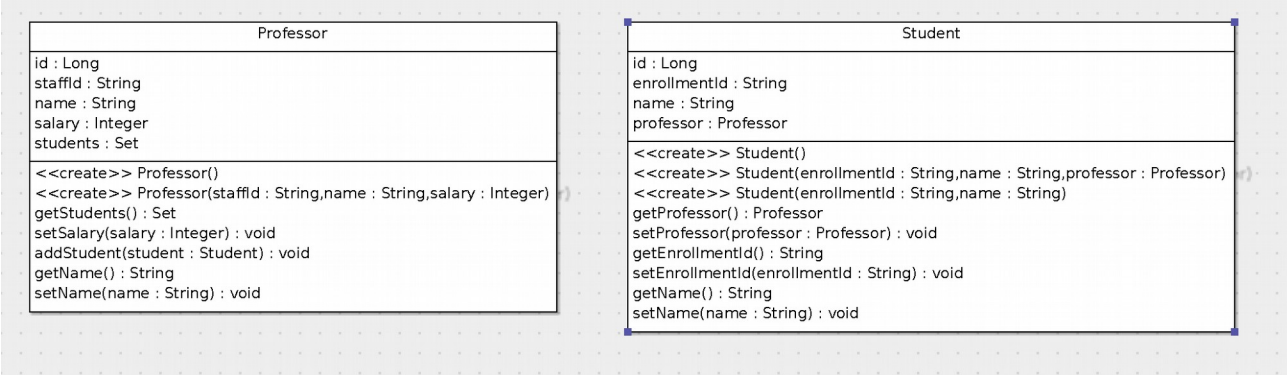


configurer le POM avec les dependencies et properties suivantes :

```
    <dependencies>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.31</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.4.2.Final</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-entitymanager</artifactId>
            <version>5.4.2.Final</version>
        </dependency>
    </dependencies>
    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>
```

Créer une classe Professor et Student

comme suit.



Partie 1 : Annotation

Annoter Professor et Student avec `@Entity, @Id, @GeneratedValue`

Dans Professor, annoter le champ student comme suit :

```
@OneToMany(mappedBy="professor")
private Set<Student> students = new HashSet<Student>();
```

Dans Student annoter le champ professor

```
@ManyToOne
@JoinColumn(name="professor_id")
private Professor professor;
```

Partie 2 : Persiter

Créer une classe TestClientPersist avec une méthode main dans le package org.formation.client

préparer le code pour la mise à jour

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

    EntityManagerFactory emf = Persistence.createEntityManagerFactory("my-pu");
    EntityManager em = emf.createEntityManager();
    EntityTransaction txn = em.getTransaction();
    try {
        txn.begin();


        txn.commit();
    } catch (Exception e) {
        if (txn != null) {
            txn.rollback();
        }
        e.printStackTrace();
    } finally {
        if(em != null) { em.close(); }
        if(emf != null) { emf.close(); }
    }
}
```

Créer, sans les associer une instance de Professor et de Student

Persister chaque instance individuellement.

Tester et vérifier en base la création des tables et de l'enregistrement

Partie 3 : Associer Student et Professor

Sur le code précédent ajouter l'étudiant au set du professeur (`getStudents().add`).

Relancer le programme.

Vérifier en base.

La clé étrangère du professeur dans student est-elle présente ?

Maintenant, faisont l'inverse associont le professeur à l'étudiant (setProfessor). Relancer le programme.

La clé étrangère est bien mise à jour.

Pourquoi ?

Partie 4 :Cascade

Dans Student ajouter la notion de cascade

```
@ManyToOne(cascade={CascadeType.PERSIST})
@JoinColumn(name="professor_id")
private Professor professor;
```

Créer, une instance de Professor et de Student.

Ajouter le professeur à l'étudiant

Persister uniquement l'étudiant.

Vérifier que les deux soit bien créés en base.

Mettre un cascade persit également dans Professor

```
@OneToMany(mappedBy="professor", cascade= {CascadeType.PERSIST})
private Set<Student> students = new HashSet<Student>();
```

Il sera maintenant plus simple de créer beaucoup d'étudiant, de les associer au professeur sans avoir à les persister un par un. Il suffit de persister le prof.

La clé étrangère du professeur dans student est-elle présente ?

Partie 4 :Mise à jour depuis l'inverse end

Professor n'est pas propriétaire de la relation, il n'a 'pas le droit' de modifier la table student.

Dans Professor ajouter une méthode addStudent(Student student)

Cette méthode doit setter le professor dans student et ajouter le student dans le set de Professor

```java
public void addStudent(Student student) {
    students.add(student);
    student.setProfessor(this);
}
```

maintenant relancer TestClient

Les clé étrangères sont à jour

Partie 5 : Lazy loading

Lister les étudiant dans l'entity manager

Vérifier qu'il y a bien en base un prof associé à plusieurs étudiants

Faire un find pour récupérer le prof.

Sans fermer l'entity manager, vérifier qu'il y a bien les étudiant dans le set du professeur.

Lister les étudiant en dehors de l'entity manager

Faire un find pour récupérer le prof.

Lister les Student du set du professeur en dehors de la session.

Par exemple comme ceci :

```java
EntityManagerFactory emf = Persistence.createEntityManagerFactory("my-pu");
EntityManager em = emf.createEntityManager();
EntityTransaction txn = em.getTransaction();
Professor prof1 = null;
try {
    txn.begin();

     prof1 = em.find(Professor.class, 13L);
    txn.commit();
} catch (Exception e) {
    if (txn != null) {
        txn.rollback();
    }
    e.printStackTrace();
} finally {
    if(em != null) { em.close(); }
}
System.out.println(prof1.getStudents());
```

On obtient l'erreur suivante :

```
Exception in thread "main" org.hibernate.LazyInitializationException: failed to lazily initialize a collection of role: org.formation.hibernate.entity.Professor.students, could not initializ
        at org.hibernate.collection.internal.AbstractPersistentCollection.throwLazyInitializationException(AbstractPersistentCollection.java:575)
        at org.hibernate.collection.internal.AbstractPersistentCollection.withTemporarySessionIfNeeded(AbstractPersistentCollection.java:214)
        at org.hibernate.collection.internal.AbstractPersistentCollection.initialize(AbstractPersistentCollection.java:554)
        at org.hibernate.collection.internal.AbstractPersistentCollection.read(AbstractPersistentCollection.java:142)
        at org.hibernate.collection.internal.PersistentSet.toString(PersistentSet.java:316)
        at java.base/java.lang.String.valueOf(String.java:2951)
        at java.base/java.io.PrintStream.println(PrintStream.java:897)
        at org.formation.hibernate.client.TestClientFetch.main(TestClientFetch.java:36)
```

Une solution classique : accéder à la liste pendant la session

```
EntityTransaction txn = em.getTransaction();
Professor prof1 = null;
try {
    txn.begin();

     prof1 = em.find(Professor.class, 13L);
     Set<Student> students = prof1.getStudents();
     students.size();
    txn.commit();
} catch (Exception e) {
    if (txn != null) {
        txn.rollback();
    }
    e.printStackTrace();
} finally {
    if(em != null) { em.close(); }
}
System.out.println(prof1.getStudents());
```

Bravo !