Association OneToMany bidirectionnelle

# Démonstration 8 du module 2

|  |
| --- |
| Les objectifs de cette démonstration :   * Déclaration d’une association OneToMany bidirectionnelle * Utilisation de l’annotation @ManyToOne (mappedBy, cascade, orphanRemoval) * Utilisation de l’annotation @JoinColumn |

Déroulement

# Contexte

* Continuer l’application précédente
* Transformons l’association unidirectionnelle entre Promo et EtudiantEni en une association bidirectionnelle

# Modification de la classe Promo :

* Ajout du paramètre mappedBy à l’annotation @0neToMany
* Et mise en commentaire de @JoinColumn car c’est l’autre côté de la relation qui est privilégié

…

@EqualsAndHashCode.Exclude

@OneToMany(mappedBy = "promo",

cascade = CascadeType.***ALL***,

orphanRemoval = **true**,

fetch = FetchType.***LAZY***)

//@JoinColumn(name="STUDENT\_ID")

**private** @Builder.Default List<EtudiantEni> etudiants = **new** ArrayList<>();

…

# Modification de la classe EtudiantEni :

* Ajout d’un attribut de type Promo annoté @ManyToOne
  + Ajout d’un nom pour la colonne de jointure
* Et les paramètres de cascade et fetch
* L’exclure de la méthode toString pour éviter la récursivité

…

//Association Bidirectionnelle avec Promo

@ManyToOne(cascade = CascadeType.***ALL***, fetch = FetchType.***LAZY***)

@JoinColumn(name = "CLASS\_ID")

@ToString.Exclude

**private** Promo promo;

}

# Tests de validation :

* Il est important de valider l’ajout, la suppression en cascade à partir de l’entité Promo
* Créer une classe de tests TestOneToManyBi

**package** fr.eni.demo.association;

**import** **static** org.assertj.core.api.Assertions.*assertThat*;

**import** **static** org.junit.jupiter.api.Assertions.*assertNull*;

**import** java.util.\*;

**import** java.util.stream.Collectors;

**import** org.junit.jupiter.api.Test;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.boot.test.autoconfigure.orm.jpa.\*;

**import** fr.eni.demo.bo.stagiaire.\*;

**import** fr.eni.demo.dal.PromoRepository;

**import** lombok.extern.slf4j.Slf4j;

@Slf4j

@DataJpaTest

**public** **class** TestOneToManyBi {

@Autowired

**private** TestEntityManager entityManager;

@Autowired

PromoRepository promoRepository;

@Test

**public** **void** test\_save() {

**final** Promo promo = Promo

.*builder*()

.nom("EDWM\_TEST")

.build();

// Association Bidirectionnelle

List<EtudiantEni> etudiants = jeuDeDonnees();

promo.setEtudiants(jeuDeDonnees());

etudiants.forEach(etudiant -> {

etudiant.setPromo(promo);

});

// Appel du comportement

**final** Promo promoDB = promoRepository.save(promo);

// Vérification de l'identifiant

*assertThat*(promoDB.getId()).isGreaterThan(0);

// Vérification de la cascade de l'association

*assertThat*(promoDB.getEtudiants()).isNotNull();

*assertThat*(promoDB.getEtudiants()).isNotEmpty();

*assertThat*(promoDB.getEtudiants().size()).isEqualTo(3);

***log***.info(promoDB.toString());

}

@Test

**public** **void** test\_delete() {

**final** Promo promo = Promo

.*builder*()

.nom("EDWM\_TEST")

.build();

// Association OneToMany Bidirectionnelle

**final** List<EtudiantEni> etudiants = jeuDeDonnees();

promo.setEtudiants(etudiants);

etudiants.forEach(etudiant -> {

etudiant.setPromo(promo);

});

// Contexte de la DB

**final** Promo promoDB = entityManager.persist(promo);

entityManager.flush();

*assertThat*(promoDB.getId()).isGreaterThan(0);

*assertThat*(promoDB.getEtudiants()).isNotNull();

*assertThat*(promoDB.getEtudiants()).isNotEmpty();

List<EtudiantEni> etudiantsDB = promoDB.getEtudiants();

List<String> listeIdEtudiantEniDB = etudiantsDB

.stream()

.map(EtudiantEni::getImmatriculation)

.collect(Collectors.*toList*());

// Appel du comportement

promoRepository.delete(promoDB);

// Vérification que l'entité a été supprimée

**final** Promo promoDB2 = entityManager.find(Promo.**class**, promo.getId());

*assertNull*(promoDB2);

// Vérifier que tous les EtudiantEni sont supprimés par cascade

*assertThat*(listeIdEtudiantEniDB).isNotNull();

*assertThat*(listeIdEtudiantEniDB).isNotEmpty();

listeIdEtudiantEniDB.forEach(etudiantId -> {

*assertThat*(etudiantId).isNotNull();

EtudiantEni etudiantDB = entityManager.find(EtudiantEni.**class**, etudiantId);

*assertNull*(etudiantDB);

});

}

@Test

**public** **void** test\_orphanRemoval() {

**final** Promo promo = Promo

.*builder*()

.nom("EDWM\_TEST")

.build();

// Association OneToMany Bidirectionnelle

**final** List<EtudiantEni> etudiants = jeuDeDonnees();

promo.setEtudiants(etudiants);

etudiants.forEach(etudiant -> {

etudiant.setPromo(promo);

});

// Contexte de la DB

**final** Promo promoDB = entityManager.persist(promo);

entityManager.flush();

*assertThat*(promoDB.getId()).isGreaterThan(0);

*assertThat*(promoDB.getEtudiants()).isNotNull();

*assertThat*(promoDB.getEtudiants()).isNotEmpty();

List<EtudiantEni> etudiantsDB = promoDB.getEtudiants();

List<String> listeIdEtudiantEniDB = etudiantsDB

.stream()

.map(EtudiantEni::getImmatriculation)

.collect(Collectors.*toList*());

// Détacher les employés de leur promo

promo.getEtudiants().clear();

// Appel du comportement

promoRepository.delete(promo);

// Vérification que l'entité a été supprimée

**final** Promo promoDB2 = entityManager.find(Promo.**class**, promo.getId());

*assertNull*(promoDB2);

// Vérifier que tous les EtudiantEni sont supprimés par orphanRemoval

*assertThat*(listeIdEtudiantEniDB).isNotNull();

*assertThat*(listeIdEtudiantEniDB).isNotEmpty();

listeIdEtudiantEniDB.forEach(etudiantId -> {

*assertThat*(etudiantId).isNotNull();

EtudiantEni etudiantDB = entityManager.find(EtudiantEni.**class**, etudiantId);

*assertNull*(etudiantDB);

});

}

**private** List<EtudiantEni> jeuDeDonnees() {

**final** List<EtudiantEni> etudiants = **new** ArrayList<>();

String immatriculation = "ENI\_CAMPUS\_20231187";

**for** (**int** i = 1; i < 4; i++) {

**final** DonneesPerso donneesPerso = DonneesPerso

.*builder*()

.nom("Nom" + i)

.prenom("Prenom" + i)

.numDom("02XXXXXXXX")

.numPortable("07XXXXXXXX")

.emailPersonnel("Nom" + i + ".Prenom" + i + "@perso.fr")

.build();

**final** EtudiantEni etudiant = EtudiantEni

.*builder*()

.immatriculation(immatriculation + i)

.email("pnom" + i + "@campus-eni.fr")

.build();

etudiant.setDonneesPerso(donneesPerso);

donneesPerso.setEtudiantEni(etudiant);

etudiants.add(etudiant);

}

**return** etudiants;

}

}

* Les tests doivent tous passés à vert

Il est important de noter, la complexité du code pour réaliser l’association bidirectionnelle

Il faut ajouter une boucle pour associer à chaque EtudiantEni sa Promo

Il faudra faire très attention à cela dans la couche métier (BLL)