Association OneToOne bidirectionnelle

# Démonstration 6 du module 2

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

Déroulement

# Contexte

* Continuer l’application précédente

Nos étudiants ont des informations privées, nous voulons les gérer dans une classe séparée ; ainsi elles pourront être anonymisées si besoin (RGPD)

* Création d’une association bidirectionnelle entre EtudiantEni et DonneesPerso
* Immatriculation de l’étudiant est unique, il est la clef primaire
* Voici le diagramme des classes :

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

# Création de la classe DonneesPerso :

* Dans le package fr.eni.bo.stagiaire
* Ajout d’un attribut de type EtudiantEni
  + Ajout de l’annotation @OneToOne
  + Et le paramètre mappedBy pour préciser qu’elle est l’entité secondaire
  + Ajout de l’annotation @JoinColumn pour préciser le nom de la colonne en base

**package** fr.eni.demo.bo.stagiaire;

**import** jakarta.persistence.\*;

**import** lombok.\*;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

@Entity

@Table(name = "STUDENT\_DATA")

**public** **class** DonneesPerso {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** id;

@Column(name = "LAST\_NAME", nullable = **false**, length = 90)

**private** String nom;

@Column(name = "FIRST\_NAME", nullable = **false**, length = 150)

**private** String prenom;

@Column(name = "HOME\_PHONE\_NUMBER", length = 12)

**private** String numDom;

@Column(name = "CELL\_NUMBER", length = 12)

**private** String numPortable;

@Column(name="PERSONAL\_EMAIL", nullable = **false**, unique = **true**, length = 255)

**private** String emailPersonnel;

// Association bidirectionnelle

@OneToOne(mappedBy = "donneesPerso")

@JoinColumn(name = "STUDENT\_ID")

**private** EtudiantEni etudiantEni;

}

# Création de l’entité EtudiantEni

* Dans le package fr.eni.bo.stagiaire
* Ajout d’un attribut de type DonneesPerso
  + Ajout de l’annotation @OneToOne avec
    - Le paramètre cascade à ALL
    - Le paramètre orphanRemoval à True 🡪 pour supprimer les données personnelles si elles deviennent orphelines
    - Le paramètre fetch à EAGER 🡪 pour charger les 2 entités en même temps
  + Ajout de l’annotation @JoinColumn pour préciser le nom de la colonne en base

**package** fr.eni.demo.bo.stagiaire;

**import** jakarta.persistence.\*;

**import** lombok.\*;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

@Entity

@Table(name = "ENI\_STUDENT")

**public** **class** EtudiantEni {

@Id

@Column(name = "STUDENT\_REGISTRATION", nullable = **false**, unique = **true**, length = 100)

**private** String immatriculation;

@Column(nullable = **false**, unique = **true**, length = 255)

**private** String email;

// Association bidirectionnelle

@OneToOne(cascade = CascadeType.***ALL***, orphanRemoval = **true**, fetch = FetchType.***EAGER***)

@JoinColumn(name = "DATA\_ID")

**private** DonneesPerso donneesPerso;

}

# Créer les Repository

**package** fr.eni.demo.dal;

**import** org.springframework.data.jpa.repository.JpaRepository;

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

**public** **interface** DonneesPersoRepository **extends** JpaRepository<DonneesPerso, Integer> {

}

**package** fr.eni.demo.dal;

**import** org.springframework.data.jpa.repository.JpaRepository;

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

**public** **interface** EtudiantEniRepository **extends** JpaRepository<EtudiantEni, String> {

}

# Tests en TDD :

* Nous allons de nouveau utiliser la logique de développement en TDD pour voir les problèmes liés aux associations bidirectionnelles
  + Et voir comment les régler
* Dupliquer la classe de tests unitaires TestOneToOneUni précédente en TestOneToOneBi
  + Ajouter l’association bidirectionnelle dans le test de persistence (test\_save)

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

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

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

**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.\*;

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

@Slf4j

@DataJpaTest

**public** **class** TestOneToOneBi {

@Autowired

**private** TestEntityManager entityManager;

@Autowired

EtudiantEniRepository repository;

@Autowired

DonneesPersoRepository donneesPersoRepository;

@Test

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

**final** String immatriculation = "ENI\_CAMPUS\_202311872";

**final** DonneesPerso donneesPerso = DonneesPerso

.*builder*()

.nom("SOPRANO")

.prenom("Camille")

.numDom("02XXXXXXXX")

.numPortable("07XXXXXXXX")

.emailPersonnel("camille.soprano@perso.fr")

.build();

**final** EtudiantEni etudiant = EtudiantEni

.*builder*()

.immatriculation(immatriculation)

.email("csoprano@campus-eni.fr")

.build();

// Association bidirectionnelle - il faut associer des 2 côtés

etudiant.setDonneesPerso(donneesPerso);

donneesPerso.setEtudiantEni(etudiant);

// Appel du comportement

**final** EtudiantEni etudiantDB = etudiantEniRepository.save(etudiant);

//log.info(etudiant.toString());

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

// Vérification de l'identifiant

*assertThat*(etudiantDB.getImmatriculation()).isEqualTo(immatriculation);

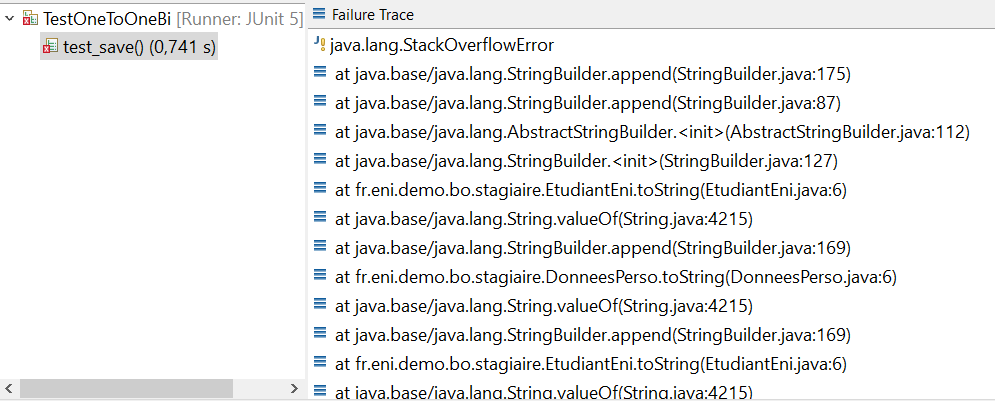
// Vérification de la cascade de l'association

*assertThat*(etudiantDB.getDonneesPerso().getId()).isGreaterThan(0);

}

}

* La ou les lignes correspondent dans nos tests aux associations uni ou bi directionnelles seront à ajouter dans les méthodes des services (couche BLL)
* Exécuter uniquement test\_save
  + Il est en erreur, il indique un StackOverflowError



* En regardant dans les traces, il faut constater qu’il appelle toString de DonneesPerso puis toString d’EtudiantEni puis de nouveau toString de DonneesPerso
  + Et cela jusqu’au StackOverflowError
* Effectivement, dès qu’il y a une association bidirectionnelle, il faut penser à se prémunir de ce problème
  + Au niveau de l’ORM on utilise : mappedBy
  + Au niveau de Java, il faut réfléchir aux méthodes qui s’appellent mutuellement :
    - equals
    - toString, …

## Prémunir du StackOverflowError

Avec Lombok, il est possible de préciser les attributs pour la redéfinition du toString et equals. Il est possible aussi de supprimer des attributs en utilisant :

* @ToString.Exclude
* @EqualsAndHashCode.Exclude

Il est possible de réfléchir de la même manière que l’ORM. Déterminer une entité prioritaire et dans l’autre sens exclure l’association

* Prenons l’EtudiantEni comme entité prioritaire.
* C’est dans DonneesPerso qu’il faut donc poser les 2 exclusions sur l’attribut de l’association

**package** fr.eni.demo.bo.stagiaire;

**import** jakarta.persistence.\*;

**import** lombok.\*;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

@Entity

@Table(name = "STUDENT\_DATA")

**public** **class** DonneesPerso {

…

// Association bidirectionnelle

@OneToOne(mappedBy = "donneesPerso")

@JoinColumn(name = "STUDENT\_ID")

**@ToString.Exclude**

**@EqualsAndHashCode.Exclude**

**private** EtudiantEni etudiantEni;

}

* Relancer le test, il passe

## Test sauvegarde entité secondaire

Vérifions que si nous enregistrons l’entité secondaire (DonneesPerso) cela enregistrera bien l’entité principale EtudiantEni

* Dupliquer test\_save en test\_save\_donneesPerso
  + Ajouter DonneesPersoRepository comme bean de la classe de tests unitaires
  + Modifier l’entité sauvegardée, c’est DonneesPerso qui est sauvegardée via DonneesPersoRepository

…

@Slf4j

@DataJpaTest

**public** **class** TestOneToOneBi {

…

@Autowired

DonneesPersoRepository donneesPersoRepository;

…

@Test

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

**final** String immatriculation = "ENI\_CAMPUS\_202311872";

**final** DonneesPerso donneesPerso = DonneesPerso

.*builder*()

.nom("SOPRANO")

.prenom("Camille")

.numDom("02XXXXXXXX")

.numPortable("07XXXXXXXX")

.emailPersonnel("camille.soprano@perso.fr")

.build();

**final** EtudiantEni etudiant = EtudiantEni

.*builder*()

.immatriculation(immatriculation)

.email("csoprano@campus-eni.fr")

.build();

// Association bidirectionnelle - il faut associer des 2 côtés

etudiant.setDonneesPerso(donneesPerso);

donneesPerso.setEtudiantEni(etudiant);

// Appel du comportement

**final** DonneesPerso donneesPersoDB = donneesPersoRepository.save(donneesPerso);

// Vérification de l'identifiant

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

// Vérification de la cascade de l'association

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

*assertThat*(etudiantDB).isNotNull();

*assertThat*(etudiantDB.getImmatriculation()).isEqualTo(immatriculation);

// Trace l'entité principale

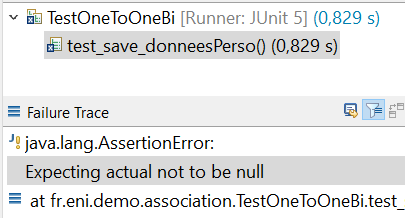
***log***.info(etudiant.toString());

}

…

}

* Exécuter test\_save\_DonneesPerso
  + Il est en échec, il a sauvegardé DonneesPerso mais pas l’EtudiantEni associé



* Ajouter le paramètre cascade à ALL pour conserver le cycle de vie identique entre les 2 entités

**package** fr.eni.demo.bo.stagiaire;

**import** jakarta.persistence.\*;

**import** lombok.\*;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

@Entity

@Table(name = "STUDENT\_DATA")

**public** **class** DonneesPerso {

…

// Association bidirectionnelle

@OneToOne(mappedBy = "donneesPerso"**, cascade = CascadeType.*ALL*)**

@JoinColumn(name = "STUDENT\_ID")

@ToString.Exclude

@EqualsAndHashCode.Exclude

**private** EtudiantEni etudiantEni;

}

* Même si on travaille depuis DonneesPerso vers EtudiantEni
  + Il faut préciser des 2 côtés le comportement de la cascade si on veut qu’elle s’applique
* Relancer le test : il est vert à présent

## Test suppression de l’entité principale

* Essayer de supprimée l’entité principale
  + Vérifier que cela supprimer l’entité associée

…

@Test

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

**final** String immatriculation = "ENI\_CAMPUS\_202311872";

**final** DonneesPerso donneesPerso = DonneesPerso

.*builder*()

.nom("SOPRANO")

.prenom("Camille")

.numDom("02XXXXXXXX")

.numPortable("07XXXXXXXX")

.emailPersonnel("camille.soprano@perso.fr")

.build();

**final** EtudiantEni etudiant = EtudiantEni

.*builder*()

.immatriculation(immatriculation)

.email("csoprano@campus-eni.fr")

.build();

// Association bidirectionnelle - il faut associer des 2 côtés

etudiant.setDonneesPerso(donneesPerso);

donneesPerso.setEtudiantEni(etudiant);

// Contexte de la DB

entityManager.persist(etudiant);

entityManager.flush();

// Appel du comportement

etudiantEniRepository.delete(etudiant);

// Vérification que l'entité a été supprimée

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

*assertNull*(etudiantDB);

DonneesPerso donneesPersoDB = entityManager.find(DonneesPerso.**class**, donneesPerso.getId());

*assertNull*(donneesPersoDB);

}

…

* Lancer le test
  + Il est vert
  + Nous avons bien mis en place la cascade dans les 2 sens donc il a retiré les 2 entités en base

## Test suppression de l’entité secondaire

Vérifions que si nous supprimons l’entité secondaire (DonneesPerso) cela supprimera bien l’entité principale EtudiantEni

* Dupliquer test\_delete en test\_delete\_donneesPerso
  + Modifier l’entité supprimée, c’est DonneesPerso qui est supprimée via DonneesPersoRepository

…

@Test

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

**final** String immatriculation = "ENI\_CAMPUS\_202311872";

**final** DonneesPerso donneesPerso = DonneesPerso

.*builder*()

.nom("SOPRANO")

.prenom("Camille")

.numDom("02XXXXXXXX")

.numPortable("07XXXXXXXX")

.emailPersonnel("camille.soprano@perso.fr")

.build();

**final** EtudiantEni etudiant = EtudiantEni

.*builder*()

.immatriculation(immatriculation)

.email("csoprano@campus-eni.fr")

.build();

// Association bidirectionnelle - il faut associer des 2 côtés

etudiant.setDonneesPerso(donneesPerso);

donneesPerso.setEtudiantEni(etudiant);

// Contexte de la DB

entityManager.persist(etudiant);

entityManager.flush();

// Appel du comportement

donneesPersoRepository.delete(donneesPerso);

// Vérification que l'entité a été supprimée

DonneesPerso donneesPersoDB = entityManager.find(DonneesPerso.**class**, donneesPerso.getId());

*assertNull*(donneesPersoDB);

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

*assertNull*(etudiantDB);

}

…

* Lancer le test
  + Il est vert
  + Le paramètre cascade à ALL sur l’attribut d’association de l’entité DonneesPerso permet de valider le comportement

## Test orphanRemoval de l’entité principale

* Créer un test pour valider le comportement d’orphelin

…

@Test

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

**final** String immatriculation = "ENI\_CAMPUS\_202311872";

**final** DonneesPerso donneesPerso = DonneesPerso

.*builder*()

.nom("SOPRANO")

.prenom("Camille")

.numDom("02XXXXXXXX")

.numPortable("07XXXXXXXX")

.emailPersonnel("camille.soprano@perso.fr")

.build();

**final** EtudiantEni etudiant = EtudiantEni

.*builder*()

.immatriculation(immatriculation)

.email("csoprano@campus-eni.fr")

.build();

// Association

etudiant.setDonneesPerso(donneesPerso);

donneesPerso.setEtudiantEni(etudiant);

// Contexte de la DB

entityManager.persist(etudiant);

entityManager.flush();

// Supprimer le lien entre l'entité EtudiantEni et l'entité DonneesPerso

etudiant.setDonneesPerso(**null**);

// Appel du comportement

etudiantEniRepository.delete(etudiant);

// Vérification que l'entité a été supprimée

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

*assertNull*(etudiantDB);

DonneesPerso donneesPersoDB = entityManager.find(DonneesPerso.**class**, donneesPerso.getId());

*assertNull*(donneesPersoDB);

}

…

* Lancer le test
  + Il est vert
  + Nous avons bien mis en place le paramètre orphanRemoval sur l’attribut DonneesPerso de l’entité

## Test orphanRemoval de l’entité secondaire

Vérifions que si nous supprimons l’entité secondaire (DonneesPerso) cela supprimera bien l’entité principale EtudiantEni en ayant mis setEtudiantEni(null)

* Dupliquer test\_orphanRemoval en test\_orphanRemoval\_DonneesPerso
  + Modifier l’entité supprimée, c’est DonneesPerso qui est supprimée via DonneesPersoRepository
  + Et mettre DonneesPerso.setEtudiantEni(null)

@Test

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

**final** DonneesPerso DonneesPerso = DonneesPerso

.*builder*()

.rue("15 rue de Paris")

.codePostal("35000")

.ville("Rennes")

.build();

**final** EtudiantEni EtudiantEni = EtudiantEni

.*builder*()

.nom("BAILLE")

.prenom("Anne-Lise")

.email("abaille@campus-eni.fr")

.immatriculation("ENI\_ECOLE\_12398")

.numDom("02XXXXXXXX")

.build();

// Association

EtudiantEni.setDonneesPerso(DonneesPerso);

DonneesPerso.setEtudiantEni(EtudiantEni);

// Contexte de la DB

entityManager.persist(EtudiantEni);

entityManager.flush();

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

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

// Supprimer le lien entre l'entité DonneesPerso et l'entité EtudiantEni

DonneesPerso.setEtudiantEni(**null**);

// Appel du comportement

DonneesPersoRepository.delete(DonneesPerso);

// Vérification que l'entité a été supprimée

DonneesPerso DonneesPersoDB = entityManager.find(DonneesPerso.**class**, DonneesPerso.getId());

*assertNull*(DonneesPersoDB);

EtudiantEni EtudiantEniDB = entityManager.find(EtudiantEni.**class**, EtudiantEni.getId());

*assertNull*(EtudiantEniDB);

}

* Exécuter test\_orphanRemoval\_DonneesPerso
  + Il est en échec, il a supprimé DonneesPerso mais pas l’EtudiantEni associé
* Ajouter le paramètre orphanRemoval à true pour conserver le cycle de vie identique entre les 2 entités

**package** fr.eni.demo.bo.stagiaire;

**import** jakarta.persistence.\*;

**import** lombok.\*;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

@Entity

@Table(name = "STUDENT\_DATA")

**public** **class** DonneesPerso {

…

// Association bidirectionnelle

@OneToOne(mappedBy = "donneesPerso", cascade = CascadeType.*ALL* **, orphanRemoval = true)**

@JoinColumn(name = "STUDENT\_ID")

@ToString.Exclude

@EqualsAndHashCode.Exclude

**private** EtudiantEni etudiantEni;

}

* Relancer le test
  + Il est vert
  + Le paramètre orphanRemoval doit être placé des 2 côtés de l’association pour s’appliquer

Remarque : une association OneToOne doit être en EAGER par défaut pour bien fonctionner