

Rappel Séance 4-1

(Mardi 01/02/2022)

En utilisant JPA, implémenter les CRUD de l'entity Facture : id (long), amount (double)

- ☐ Créer l'Entity Facture avec le mapping ORM JPA
- ☐ Créer les fichiers de configuration nécessaires
- ☐ Créer les méthodes CRUD dans la classe d'implémentation du package DAO.

Objectifs de la séance 4-2

(Jeudi 03/02/2022)

Les objectifs de la séance d'aujourd'hui:

- Objectif 4.1** : Implémenter les associations **@ManyToMany** entre les entités **JPA**
- Objectif 4.2** : Implémenter les associations **@OneToMany** entre les entités **JPA**
- Objectif 4.3** : Implémenter les associations **@ManyToMany** entre les entités **JPA**
- Objectif 4.4** : Implémenter les associations **@OneToOne** entre les entités **JPA**
- Objectif 4.5** : 2 Fetch types : EAGER et LAZY
- Objectif 4.6** : 6 cascade types : persist, remove, refresh, merge, detach, all
- Objectif 4.7** : Mapping d'une clé composé en utilisant JPA annotation

Complément Youtube de cette séance: <https://youtu.be/bFcWhKagARk>

Dans ce TP on suppose que :

- ☒ ~~Vous avez réalisé totalement le TP3~~

SI CE N'EST PAS LE CAS : FAIRE D'ABORD LE TP3 D'URGENCE

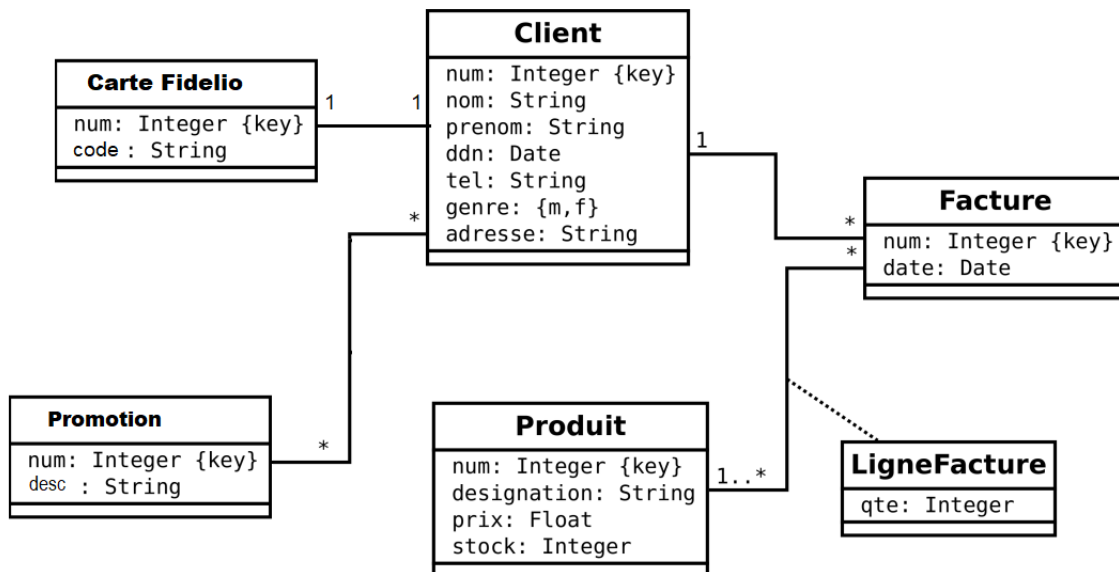
LE RÉSULTAT DU TP3 EST LE SUIVANT:

```

Hibernate:
    drop table if exists Client
Hibernate:
    create table Client (
        id bigint not null auto_increment,
        name varchar(255),
        primary key (id)
    )
Jan 13, 2022 5:28:14 PM org.hibernate.tool.hbm2ddl.SchemaExport execute
INFO: HHH000230: Schema export complete
creation bean dao
Hibernate:
    insert
    into
        Client
        (name)
    values
        (?)
    
```

INTRODUCTION

Supposons maintenant que nous avons le diagramme MCD suivant à implémenter en JPA.



I. Les association JPA:

@OneToMany : One Client To Many Factures

@ManyToOne : Many Factures To One Client

1. CRÉER L'ENTITY FACTURE DANS LE PACKAGE MODELS

```

package models;

import lombok.Getter;
import lombok.Setter;
import lombok.ToString;
    
```

```
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
@Getter
@Setter
@ToString
public class Facture {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String amount;
    private String description;

    public Facture() {
    }

    public String getAmount() {
        return amount;
    }
}
```

2. MODIFIER L'ENTITÉ CLIENT EN AJOUTANT LE MAPPING DE L'ASSOCIATION @OneToMany

```
package models;

import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

import javax.persistence.*;
import java.util.List;

@Entity
@ToString
public class Client {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    private String name;

    @OneToMany
    private List<Facture> factures;

    public Client() {
    }
    public Client(String name) {
        this.name = name;
    }
}
```

3. MODIFIER LA CLASSE DE DÉMARRAGE APPLICATIONRUNNER POUR AJOUTER UN CLIENT AVEC DES FACTURES.

```
import models.Client;
import models.Facture;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import presentation.ClientController;

import java.util.Arrays;
import java.util.List;

public class ApplicationRunner {

    public static void main(String[] args) {
        ApplicationContext ctx=new ClassPathXmlApplicationContext("spring.xml");
        ClientController ctr= (ClientController) ctx.getBean("ctrl1");

        Client client = new Client("OMAR");
        List<Facture> factures = Arrays.asList(new Facture(1500.0,"facture1"),new
Facture(2000.0,"facture2"));
        client.setFactures(factures);

        ctr.save(client);
    }
}
```

4. EXÉCUTER LA CLASSE DE DÉMARRAGE APPLICATIONRUNNER ET REMARQUER L'ERREUR

```
Caused by:
com.mysql.jdbc.exceptions.jdbc4.MySQLIntegrityConstraintViolationException:
Cannot add or update a child row: a foreign key constraint fails
(`mon_pfe/client_facture`, CONSTRAINT `FK7kiljgn5hfiqfkwpnvpivs7dl` FOREIGN KEY
(`factures_id`) REFERENCES `facture` (`id`))
    at
    java.base/jdk.internal.reflect.NativeConstructorAccessorImpl.newInstance0(Native
Method)
    at
    java.base/jdk.internal.reflect.NativeConstructorAccessorImpl.newInstance(NativeCo
nstructorAccessorImpl.java:62)
    at
    java.base/jdk.internal.reflect.DelegatingConstructorAccessorImpl.newInstance(Dele
gatingConstructorAccessorImpl.java:45)
    at
    java.base/java.lang.reflect.Constructor.newInstance(Constructor.java:490)
```

5. AJOUTER LE CASCADE PERSIST À L'ASSOCIATION @ONETO MANY AU NIVEAU DE L'ENTITY CLIENT

```
@OneToMany(cascade = {CascadeType.PERSIST})
private List<Facture> factures;
```

6. EXÉCUTER APPLICATION RUNNER ET REMARQUER QUE LES FACTURES DU CLIENT SONT AUSSI AJOUTÉES GRÂCE AU CASCADE PERSIST.

```
Hibernate: create table Client (id bigint not null auto_increment, name
varchar(255), primary key (id))
Hibernate: create table Client_Facture (Client_id bigint not null, factures_id
bigint not null)
Hibernate: create table Facture (id bigint not null auto_increment, amount double
```

```
precision not null, description varchar(255), primary key (id))
Hibernate: alter table Client_Facture add constraint UK_lr8f7qrelgqwi31ljqm6rom8n
unique (factures_id)
Hibernate: alter table Client_Facture add constraint FK7kiljgn5hfiqfkwpnpvpi7d1
foreign key (factures_id) references Facture (id)
Hibernate: alter table Client_Facture add constraint FKribyila2i35qy3f0vy8lvj8ee
foreign key (Client_id) references Client (id)
Jan 13, 2022 6:16:59 PM org.hibernate.tool.hbm2ddl.SchemaExport execute
INFO: HHH000230: Schema export complete
creation bean dao
Hibernate: insert into Client (name) values (?)
Hibernate: insert into Facture (amount, description) values (?, ?)
Hibernate: insert into Facture (amount, description) values (?, ?)
Hibernate: insert into Client_Facture (Client_id, factures_id) values (?, ?)
Hibernate: insert into Client_Facture (Client_id, factures_id) values (?, ?)
```

SAUF QUE LA SPÉCIFICATION JPA CRÉE PAR DÉFAUT UNE TABLE DE JOINTURE **Client_Facture** POUR NOTRE ASSOCIATION **@ONEToMANY**. POUR REMPLACER LA TABLE DE JOINTURE PAR UNE SIMPLE CLÉ ÉTRANGÈRE IL FAUT UTILISER L'ANNOTATION **@JOINCOLUMN** AVEC L'ASSOCIATION **@ONEToMANY**

7.

8. MODIFIER L'ENTITÉ FACTURE EN AJOUTANT UN ATTRIBUT CLIENT MAPPÉ EN UTILISANT L'ASSOCIATION **@MANYToONE** ET L'ANNOTATION **@JOINCOLUMN** . AJOUTER L'ATTRIBUT CLIENT AU CONSTRUCTEUR DE LA CLASSE FACTURE:

```
package models;

import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

import javax.persistence.*;

@Entity
@Getter
@Setter
@ToString
public class Facture {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private double amount;
    private String description;

    @ManyToOne
    @JoinColumn(name = "client_id")
    private Client client;

    public Facture() {
    }

    public Facture(double amount, String description, Client client) {
        this.amount = amount;
        this.description = description;
        this.client = client;
    }
}
```

```
}
```

9. MODIFIER L'ASSOCIATION @ONEToMANY AN NIVEAU L'ENTITÉ CLIENT EN AJOUTANT MAPPEDBY CLIENT

```
@OneToMany(cascade = {CascadeType.PERSIST}, mappedBy = "client")
private List<Facture> factures;
```

10. EXÉCUTER LA CLASSE APPLICATIONRUNNER ET VÉRIFIER LE RÉSULTAT SUIVANT :

```
Hibernate: create table Client (id bigint not null auto_increment, name
varchar(255), primary key (id))
Hibernate: create table Facture (id bigint not null auto_increment, amount double
precision not null, description varchar(255), client_id bigint, primary key (id))
Hibernate: alter table Facture add constraint FKhnasi6n05kllawhofl2f0slrj foreign
key (client_id) references Client (id)
Hibernate: insert into Client (name) values (?)
Hibernate: insert into Facture (amount, client_id, description) values (?, ?, ?)
Hibernate: insert into Facture (amount, client_id, description) values (?, ?, ?)
```

II. Les association JPA:

@ManyToMany : Many Client To Many Promotion

@OneToOne : One Client To One Carte Fedilio

11. CRÉER L'ENTITY PROMOTION AVEC LES ANNOTATIONS JPA NÉCESSAIRE

```
package models;

import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
@Getter
@Setter
@ToString
public class Promotion {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String description;

    public Promotion(String description) {
        this.description = description;
    }
}
```

12. MODIFIER L'ENTITY CLIENT EN AJOUTER L'ASSOCIATION @MANYToMANY AVEC L'ENTITY PROMOTION

Dossier des travaux pratiques. Module : JEE FRWKS .Années scolaire 2021/2022. Niveau : MASTER FST Settat

Professeur : M. Boulchahoub Hassan hboulchahoub@gmail.com

Mise à jour 1 Feb 2022

```
public class Client {
//...
@ManyToMany(cascade = {CascadeType.PERSIST})
private List<Promotion> promotions;
//...
}
```

13. MODIFIER LA CLASSE D'EXECUTION APPLICATIONRUNNER

```
public class ApplicationRunner {

    public static void main(String[] args) {
        ApplicationContext ctx=new ClassPathXmlApplicationContext("spring.xml");
        ClientController ctr= (ClientController) ctx.getBean("ctrl1");

        Client client = new Client("OMAR");
        List<Promotion> promotions=Arrays.asList(new Promotion("remise 10%"),new
        Promotion("solde 40%"));
        client.setPromotions(promotions);

        ctr.save(client);
    }
}
```

14. EXÉCUTER APPLICATION RUNNER ET REMARQUER LES REQUÊTES SQL GÉNÉRÉES

```
Hibernate: create table Client (id bigint not null auto_increment, name
varchar(255), primary key (id))
Hibernate: create table Client_Promotion (Client_id bigint not null,
promotions_id bigint not null)
Hibernate: create table Facture (id bigint not null auto_increment, amount double
precision not null, description varchar(255), client_id bigint, primary key (id))
Hibernate: create table Promotion (id bigint not null auto_increment, description
varchar(255), primary key (id))

Hibernate: alter table Client_Promotion add constraint
FKdgcylk952iehspm2x6gguibmf foreign key (promotions_id) references Promotion (id)
Hibernate: alter table Client_Promotion add constraint
FKglgj3cdt82bvrwx0mt52jyrid foreign key (Client_id) references Client (id)
Hibernate: alter table Facture add constraint FKhnasi6n05kllawhofl2f0slrj foreign
key (client_id) references Client (id)

Hibernate: insert into Client (name) values (?)
Hibernate: insert into Promotion (description) values (?)
Hibernate: insert into Promotion (description) values (?)
Hibernate: insert into Client_Promotion (Client_id, promotions_id) values (?, ?)
Hibernate: insert into Client_Promotion (Client_id, promotions_id) values (?, ?)
```

15. CHANGER LE NOM DE LA TABLE INTERMÉDIAIRE EN UTILISANT L'ANNOTATION @JOINTABLE AVEC L'ASSOCIATION @MANYTOMANY AU NIVEAU DE L'ENTITY CLIENT

```
public class Client {
//...
@ManyToMany(cascade = {CascadeType.PERSIST})
```

```
@JoinTable(name="my_join_table_client_promotion",joinColumns = @JoinColumn(
    name = "client_fk",
    referencedColumnName = "id"
),
    inverseJoinColumns = @JoinColumn(
        name = "promotion_fk",
        referencedColumnName = "id"
    ))
private List<Promotion> promotions;

//...
}
```

16. EXÉCUTER APPLICATION RUNNER ET REMARQUER LES REQUÊTES SQL GÉNÉRÉES

```
Hibernate: insert into Client (name) values (?)
Hibernate: insert into Promotion (description) values (?)
Hibernate: insert into Promotion (description) values (?)
Hibernate: insert into my_join_table_client_promotion (client_fk, promotion_fk)
values (?, ?)
Hibernate: insert into my_join_table_client_promotion (client_fk, promotion_fk)
values (?, ?)
```

MAINTENANT TESTONS L'ASSOCIATION **@ONEToOne** ENTRE LE CLIENT ET SA CARTE FIDELIO

17. CRÉER ENTITY CARTEFIDELIO AVEC LES ANNOTATIONS JPA NÉCESSAIRES

```
package models;

import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
@Getter
@Setter
@ToString
public class CarteFidelio {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String code;

    public CarteFidelio(String code) {
        this.code = code;
    }
}
```

18. MODIFIER L'ENTITÉ CARTE FIDELIO EN AJOUTANT L'ASSOCIATION **@ONEToOne** AVEC L'ENTITY CARTE CLIENT


```
@Entity
@Getter
@Setter
@ToString
public class CarteFidelio {

    //...

    @OneToOne
    @JoinColumn(name = "client_id")
    private Client client;

    //...

}
```

19. MODIFIER L'ENTITÉ CLIENT EN AJOUTANT L'ASSOCIATION @ONE_TO_ONE AVEC L'ENTITÉ CARTE CLIENT EN UTILISANT **MAPPEDBY** ATTRIBUTE

```
public class Client {
    //...
    @OneToOne(cascade = {CascadeType.PERSIST}, mappedBy = "client")
    private CarteFidelio carteFidelio;
    //...
}
```

20. MODIFIER LA CLASSE DE DÉMARRAGE APPLICATIONRUNNER COMME SUIVANT:

```
public class ApplicationRunner {

    public static void main(String[] args) {
        ApplicationContext ctx=new ClassPathXmlApplicationContext("spring.xml");
        ClientController ctr= (ClientController) ctx.getBean("ctrl1");
        Client client = new Client("OMAR");
        CarteFidelio carteFidelio = new CarteFidelio("A29930489");
        carteFidelio.setClient(client);
        client.setCarteFidelio(carteFidelio);
        ctr.save(client);
    }
}
```

21. EXÉCUTER LA CLASSE APPLICATIONRUNNER ET VÉRIFIER LES REQUÊTES SQL


```
Ati
Hibernate: create table Client (id bigint not null auto_increment, name
varchar(255), primary key (id))

Hibernate: create table CarteFidelio (id bigint not null auto_increment, code
varchar(255), client_id bigint, primary key (id))

Hibernate: alter table CarteFidelio add constraint FK804xbth7w7ruph185ykcss90x
foreign key (client_id) references Client (id)

Hibernate: insert into Client (name) values (?)
Hibernate: insert into CarteFidelio (client_id, code) values (?, ?)
```

Cas pratique

	<p>Cas pratique pour l'utilisation des associations JPA</p> <p><i>(ce cas pratique sera noté sur 5 points dans la note des TP)</i></p> <ol style="list-style-type: none"> 1) Créer l'entity Produit avec les attributs (id, name, price) 2) Implémenter l'association bidirectionnelle JPA suivante: Produit ↔ Facture 1) Créer l'entity Adresse avec les attributs (id, avenue ,ville, pays) 2) Implémenter l'association bidirectionnelle JPA suivante: Client ↔ Adresse <p><i>Les tests seront vérifié au niveau de votre machine</i></p>
---	---

III. Les Mapping d'une clé composée en JPA

1. @Embedded and @Embeddable

SUPPOSONS MAINTENANT QUE NOUS DISPOSONS DE L'ENTITY COMPANY DANS NOTRE PROJET

La table company doit contenir des informations basiques comme: company name, address, phone, et aussi des informations de la personne qui représente la company. L'Entity est la suivante:

```
package models;

import lombok.Data;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
@Data
public class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE)
    private Long id;
    private String name;
    private String address;
    private String phone;
}
```

```
private String contactFirstName;
private String contactLastName;
private String contactPhone;
}
```

La table créée est la suivante:

```
Hibernate: create table Company (id bigint not null, address
varchar(255), contactFirstName varchar(255), contactLastName
varchar(255), contactPhone varchar(255), name varchar(255), phone
varchar(255), primary key (id)) ENGINE=InnoDB
```

Il est possible d'améliorer l'entity company en créant une classe dédiée pour ContactPerson comme suivant :

```
package models;

import lombok.Data;

import javax.persistence.Embeddable;

@Embeddable
@Data
public class ContactPerson {
    private String firstName;
    private String lastName;
    private String phone;
}
```

Cette classe doit être déclarée @Embeddable

Dans la l'Entity Company, on ajoute un attribut de type ContactPerson et on le déclare @Embedded

```
package models;

import lombok.Data;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
```

```
import javax.persistence.Id;

@Entity
@Data
public class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE)
    private Long id;
    private String name;
    private String address;
    private String phone;
    @Embedded
    private ContactPerson contactPerson;
}
```

Parce que l'attribut phone existe à la fois dans Company et dans Contact Person, Hibernate lance l'exception suivante:

Caused by: org.hibernate.MappingException: Repeated column in mapping for entity: models.Company column: phone (should be mapped with insert="false" update="false")

Pour corriger cette erreur utiliser on utilise @AttributeOverride:

```
@Embedded
@AttributeOverride(name = "phone", column = @Column(name =
"PHONE_PERSON"))
private ContactPerson contactPerson;
```

La table créée est la suivante:

Hibernate: create table Company (id bigint not null, address varchar(255), firstName varchar(255), lastName varchar(255), PHONE_PERSON varchar(255), name varchar(255), phone varchar(255), primary key (id)) ENGINE=InnoDB

2. Mapping Composite Primary Key Using @IdClass Annotation

Acceptons maintenant qu'une Company dispose d'une clé primaire composée de deux attributs : RC (registre de commerce) et Id

Tribunal. Comment mapper cette clé en JPA. @Id ne fait plus l'affaire ici, car il est utilisé seulement pour mapper les clé primaire simple.

D'abord créer une classe CompanyId pour représenter la clé primaire composée.

Cette classe:

- Doit être déclarée @Embeddable*
- Doit être Serializable*
- Doit redéfinir la méthode hashCode et la méthode equals*

```
package models;

import javax.persistence.Embeddable;
import java.io.Serializable;
import java.util.Objects;

@Embeddable
public class CompanyId implements Serializable {
    private long rc;
    private long idTribunal;

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        CompanyId companyId = (CompanyId) o;
        return rc == companyId.rc && idTribunal == companyId.idTribunal;
    }

    @Override
    public int hashCode() {
        return Objects.hash(rc, idTribunal);
    }
}
```

Modifier la classe Company en utilisant l'annotation @IdClass

```
package models;

import lombok.Data;
import javax.persistence.*;

@Entity
@Data
```

```
@IdClass(CompanyId.class)
public class Company {
    @Id
    private long rc;
    @Id
    private long idTribunal;

    private String name;
    private String address;
    private String phone;
    @Embedded
    @AttributeOverride(name = "phone", column = @Column(name =
    "PHONE_PERSON"))
    private ContactPerson contactPerson;
}
```

Exécuter pour vérifier que la table Company créée contiendra
primary key (id Tribunal, rc)

```
Hibernate: create table Company (idTribunal bigint not null, rc bigint
not null, address varchar(255), firstName varchar(255), lastName
varchar(255), PHONE_PERSON varchar(255), name varchar(255), phone
varchar(255), primary key (idTribunal, rc)) ENGINE=InnoDB
```

3. Mapping Composite Primary Key Using @Embeddable and @EmbeddedId Annotations

D'abord Déclarer la classe CompanyId @Embeddable

```
@Embeddable
public class CompanyId implements Serializable {...}
```

Ensuite, modifier l'Entity Company en remplaçant @Id par @EmbeddedId

```
package models;

import lombok.Data;
import javax.persistence.*;

@Entity
@Data
public class Company {

    @EmbeddedId
```

```
private CompanyId id;  
  
private String name;  
private String address;  
private String phone;  
@Embedded  
@AttributeOverride(name = "phone", column = @Column(name =  
"PHONE_PERSON"))  
private ContactPerson contactPerson;  
}
```

Exécuter pour vérifier que la table Company créée contiendra primary key (id Tribunal, rc)

```
Hibernate: create table Company (idTribunal bigint not null, rc bigint  
not null, address varchar(255), firstName varchar(255), lastName  
varchar(255), PHONE_PERSON varchar(255), name varchar(255), phone  
varchar(255), primary key (idTribunal, rc)) ENGINE=InnoDB
```