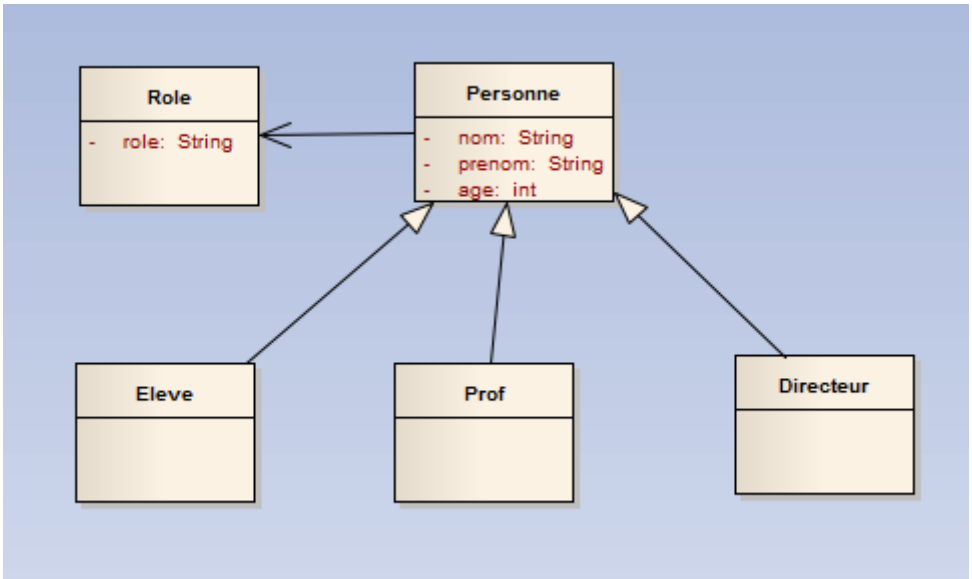
Rappel Séance 10 (week-end 29-30/01/2022)

En utilisant les association JPA, créer et mapper les liens d'héritages suivants:



Objectifs de la séance 11 (week-end 29-30/01/2022)

Les objectifs de la séance d'aujourd'hui:

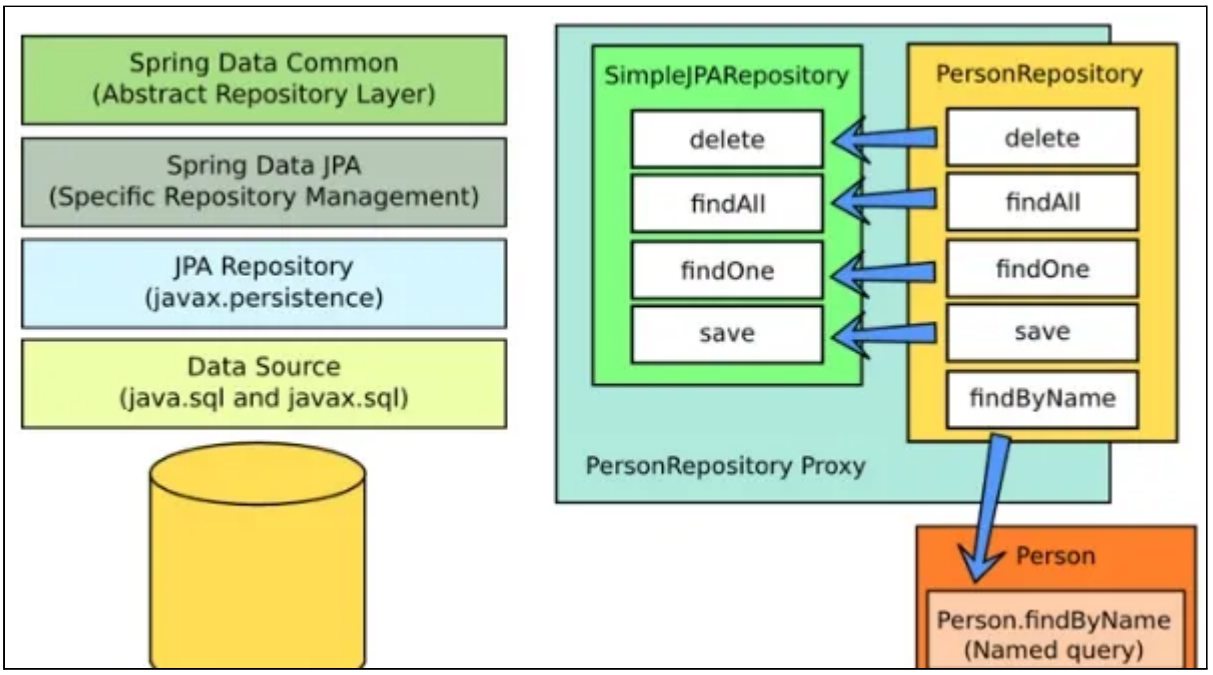
Objectif **11.1** :
Utiliser le framework **Spring Data** pour implémenter la couche DAO
Objectif **11.2** :
Mapping d'une clé primaire composée **@EmbeddedId**, **@Embeddable** et **@IdClass**

Complément Youtube de cette séance:
Dans ce TP on suppose que :

- ☑ ~~*Vous avez réalisé totalement le TP8.*~~
- ☑ ~~*Vous avez réalisé totalement le TP9.*~~
- ☑ ~~*Vous avez réalisé totalement le TP10.*~~

Si ce n'est pas le cas : Faire d'abord le TP8, TP9 et le TP10 d'urgence

Introduction



I. Utilisation de Spring Data Framework

1. Ajouter les dépendances du spring data dans le fichier pom.xml

```
<!--
https://mvnrepository.com/artifact/org.springframework.data/spring-data-jpa
-->
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>2.5.0</version>
</dependency>
```

2. Supprimer les classes d'implémentation de la couche Dao

3. Modifier les interfaces de la couche Dao en ajoutant le lien d'héritage avec l'interface spring data "CrudRepository"

```
package dao;

import models.Client;
import org.springframework.data.repository.CrudRepository;

@Repository
public interface IClientDao extends CrudRepository<Client,Long> {
}
```

4. Modifier le fichier de création des beans resources/spring.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:data="http://www.springframework.org/schema/data/jpa"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd
    http://www.springframework.org/schema/data/jpa
    http://www.springframework.org/schema/data/jpa/spring-jpa.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx.xsd">

    <data:repositories base-package="dao" />
    <context:component-scan base-package="service" />
    <context:component-scan base-package="presentation" />
    <tx:annotation-driven />

    <bean id="entityManagerFactory"
          class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <property name="persistenceUnitName" value="unit_person" />
    </bean>

    <bean id="transactionManager"
          class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory"
                  ref="entityManagerFactory" />
    </bean>
</beans>
```

5. Maintenant, puisque les méthodes de la couche Dao sont implémentées par Spring Data Framework, Compléter la couche service en ajoutant toutes les méthodes nécessaires pour la gestion d'un client.

L'interface de la couche service

```java
package service;

import models.Client;
import java.util.List;

public interface IClientService {
    Client save(Client clt);
    Client modify(Client clt);
    void remove(long idClt);
    Client getOne(long idClt);
    List<Client> getAll();
}
```

La classe d'implémentation de la couche service

```java
package service;

import dao.IClientDao;
import models.Client;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service
public class ClientServiceImpl implements IClientService {
    @Autowired
    private IClientDao dao;
    @Override
    @Transactional
    public Client save(Client clt) {
        return dao.save(clt);
    }

    @Override
    @Transactional
    public Client modify(Client newClt) {
        Client oldClt = dao.findById(newClt.getId()).get();
        oldClt.setName(newClt.getName());
        return dao.save(oldClt);
    }

    @Override
    @Transactional
    public void remove(long idClt) {
         dao.deleteById(idClt);
    }

    @Override
    public Client getOne(long idClt) {
        return dao.findById(idClt).get();
    }

    @Override
    public List<Client> getAll() {
        return (List<Client>) dao.findAll();
    }
```

```
}
```

6. Modifier le contrôleur pour appeler toutes les méthodes de la couche service.

```
package presentation;

import models.Client;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import service.IClientService;
import java.util.List;

@Controller(value = "ctrl1")
public class ClientController {
   @Autowired
   private IClientService service;

   public Client save(Client clt) {
       return service.save(clt);
   }
   public Client modify(Client clt) {
       return service.modify(clt);
   }
   public void remove(long idClt) {
        service.remove(idClt);
   }
   public Client getOne(long idClt) {
       return service.getOne(idClt);
   }
   public List<Client> getAll() {
       return service.getAll();
   }
}
```

7. Tester les méthodes du contrôleur dans la classe Application Runner

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

public class ApplicationRunner {

   public static void main(String[] args) {
       ApplicationContext ctx=new
```

```
ClassPathXmlApplicationContext("spring.xml");
        ClientController ctr= (ClientController) ctx.getBean("ctrl1");
        Client client1 = new Client("Omar");
        Client client2 = new Client("Said");
        Client client3 = new Client("Ahmed");

        // Test1 => save 3 Clients
        client1=ctr.save(client1);
        client2=ctr.save(client2);
        client3=ctr.save(client3);

        // Test2 => getAll Clients before modify and remove
        ctr.getAll().stream()
                .forEach(i-> System.out.println(i));

        // Test3 => getOne Client service
        System.out.println(ctr.getOne(1));

        // Test4 => modify Client service
        client1.setName("Hassan");
        ctr.modify(client1);

        // Test5 => remove Client service
        ctr.remove(2);
        // Test getAll Client after modify and remove
        ctr.getAll().stream()
                .forEach(i-> System.out.println(i));
    }
}
```

8. Analyser les requêtes genre par Spring Data

```
_// Test1 => save 3 Clients
Hibernate: select tbl.next_val from hibernate_sequences tbl where tbl.sequence_name=?
for update
Hibernate: insert into hibernate_sequences (sequence_name, next_val)  values (?,?)
Hibernate: update hibernate_sequences set next_val=?  where next_val=? and
sequence_name=?
Hibernate: insert into Client (name, id) values (?, ?)
Hibernate: select tbl.next_val from hibernate_sequences tbl where tbl.sequence_name=?
for update
Hibernate: update hibernate_sequences set next_val=?  where next_val=? and
sequence_name=?
Hibernate: insert into Client (name, id) values (?, ?)
Hibernate: select tbl.next_val from hibernate_sequences tbl where tbl.sequence_name=?
for update
Hibernate: update hibernate_sequences set next_val=?  where next_val=? and
```

```
sequence_name=?
Hibernate: insert into Client (name, id) values (?, ?)
Jan 27, 2022 7:40:44 PM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator
initiateService
INFO: HHH000397: Using ASTQueryTranslatorFactory
```

// Test2 => get All Clients before modify and remove

```
Hibernate: select client0_.id as id1_1_, client0_.name as name2_1_, client0_1_.status as
status1_2_, client0_2_.preferences as preferen1_3_, case when client0_1_.id is not null
then 1 when client0_2_.id is not null then 2 when client0_.id is not null then 0 end as
clazz_ from Client client0_ left outer join Normal client0_1_ on
client0_.id=client0_1_.id left outer join Vip client0_2_ on client0_.id=client0_2_.id
Hibernate: select addresses0_.FK_CLIENT_ID as FK_CLIEN3_0_0_, addresses0_.id as
id1_0_0_, addresses0_.id as id1_0_1_, addresses0_.FK_CLIENT_ID as FK_CLIEN3_0_1_,
addresses0_.description as descript2_0_1_ from Address addresses0_ where
addresses0_.FK_CLIENT_ID=?
Hibernate: select addresses0_.FK_CLIENT_ID as FK_CLIEN3_0_0_, addresses0_.id as
id1_0_0_, addresses0_.id as id1_0_1_, addresses0_.FK_CLIENT_ID as FK_CLIEN3_0_1_,
addresses0_.description as descript2_0_1_ from Address addresses0_ where
addresses0_.FK_CLIENT_ID=?
Hibernate: select addresses0_.FK_CLIENT_ID as FK_CLIEN3_0_0_, addresses0_.id as
id1_0_0_, addresses0_.id as id1_0_1_, addresses0_.FK_CLIENT_ID as FK_CLIEN3_0_1_,
addresses0_.description as descript2_0_1_ from Address addresses0_ where
addresses0_.FK_CLIENT_ID=?
Client(id=1, name=Omar, addresses=[])
Client(id=2, name=Said, addresses=[])
Client(id=3, name=Ahmed, addresses=[])
```

// Test3 => get One Client by Id

```
Hibernate: select client0_.id as id1_1_0_, client0_.name as name2_1_0_,
client0_1_.status as status1_2_0_, client0_2_.preferences as preferen1_3_0_, case when
client0_1_.id is not null then 1 when client0_2_.id is not null then 2 when client0_.id
is not null then 0 end as clazz_0_, addresses1_.FK_CLIENT_ID as FK_CLIEN3_0_1_,
addresses1_.id as id1_0_1_, addresses1_.id as id1_0_2_, addresses1_.FK_CLIENT_ID as
FK_CLIEN3_0_2_, addresses1_.description as descript2_0_2_ from Client client0_ left
outer join Normal client0_1_ on client0_.id=client0_1_.id left outer join Vip client0_2_
on client0_.id=client0_2_.id left outer join Address addresses1_ on
client0_.id=addresses1_.FK_CLIENT_ID where client0_.id=?
Client(id=1, name=Omar, addresses=[])
```

// Test4 => Modify Client

```
Hibernate: select client0_.id as id1_1_0_, client0_.name as name2_1_0_,
client0_1_.status as status1_2_0_, client0_2_.preferences as preferen1_3_0_, case when
client0_1_.id is not null then 1 when client0_2_.id is not null then 2 when client0_.id
is not null then 0 end as clazz_0_, addresses1_.FK_CLIENT_ID as FK_CLIEN3_0_1_,
addresses1_.id as id1_0_1_, addresses1_.id as id1_0_2_, addresses1_.FK_CLIENT_ID as
FK_CLIEN3_0_2_, addresses1_.description as descript2_0_2_ from Client client0_ left
outer join Normal client0_1_ on client0_.id=client0_1_.id left outer join Vip client0_2_
on client0_.id=client0_2_.id left outer join Address addresses1_ on
client0_.id=addresses1_.FK_CLIENT_ID where client0_.id=?
Hibernate: update Client set name=? where id=?
```

// Test5 => Remove Client by Id

```
Hibernate: select client0_.id as id1_1_0_, client0_.name as name2_1_0_,
client0_1_.status as status1_2_0_, client0_2_.preferences as preferen1_3_0_, case when
client0_1_.id is not null then 1 when client0_2_.id is not null then 2 when client0_.id
```

```
is not null then 0 end as clazz_0_, addresses1_.FK_CLIENT_ID as FK_CLIEN3_0_1_,
addresses1_.id as id1_0_1_, addresses1_.id as id1_0_2_, addresses1_.FK_CLIENT_ID as
FK_CLIEN3_0_2_, addresses1_.description as descript2_0_2_ from Client client0_ left
outer join Normal client0_1_ on client0_.id=client0_1_.id left outer join Vip client0_2_
on client0_.id=client0_2_.id left outer join Address addresses1_ on
client0_.id=addresses1_.FK_CLIENT_ID where client0_.id=?
Hibernate: delete from Client where id=?
                // Test6 => get All Client after removing and updating
Hibernate: select client0_.id as id1_1_, client0_.name as name2_1_, client0_1_.status as
status1_2_, client0_2_.preferences as preferen1_3_, case when client0_1_.id is not null
then 1 when client0_2_.id is not null then 2 when client0_.id is not null then 0 end as
clazz_ from Client client0_ left outer join Normal client0_1_ on
client0_.id=client0_1_.id left outer join Vip client0_2_ on client0_.id=client0_2_.id
Hibernate: select addresses0_.FK_CLIENT_ID as FK_CLIEN3_0_0_, addresses0_.id as
id1_0_0_, addresses0_.id as id1_0_1_, addresses0_.FK_CLIENT_ID as FK_CLIEN3_0_1_,
addresses0_.description as descript2_0_1_ from Address addresses0_ where
addresses0_.FK_CLIENT_ID=?
Hibernate: select addresses0_.FK_CLIENT_ID as FK_CLIEN3_0_0_, addresses0_.id as
id1_0_0_, addresses0_.id as id1_0_1_, addresses0_.FK_CLIENT_ID as FK_CLIEN3_0_1_,
addresses0_.description as descript2_0_1_ from Address addresses0_ where
addresses0_.FK_CLIENT_ID=?
Client(id=1, name=Hassan, addresses=[])
Client(id=3, name=Ahmed, addresses=[])
```

Cas pratique pour l'utilisation de Spring Data

(ce cas pratique sera noté sur 5 points dans la note des TPs)

1- Ajouter les méthodes nécessaires dans les trois couches pour trouver les clients par leur nom.

Indication : dans la couche Dao, il suffit d'ajouter la signature de la méthode suivante au niveau de l'interface ClientDao

```
List<Client> findByName(String name);
// Name est un attribut dans la classe Client
```

```
@Repository
public interface IClientDao extends
  CrudRepository<Client,Long> {
    List<Client> findByName(String name);
  }
```

2- Par analogie, Ajouter les méthodes nécessaires pour la gestion d'une autre entité à votre choix : Facture, Commande, Adresse…

II. 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;
```

```java
@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

```java
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:

a. *Doit être déclarée `@Embeddable`*
b. *Doit être `Serializable`*
c. *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
```