Rappel Séance 6 (week-end 18-19/12/2021)

Vu l'importance de la séance 6 dans cette formation, ce rappel sera fait en classe en collaboration des étudiants.

Objectifs de la séance 7 (week-end 25-26/12/2021)

Les objectifs de la séance d'aujourd'hui:

Objectif 7.1 : Création des **annotations** en Java
Objectif 7.2 : Rappel **Three Layered Architecture**
Objectif 7.3 : Rappel **Inversion Of Control, Dependency Injection**
Objectif 7.4 : Commencer à utiliser les annotations **JPA**

Complément Youtube de cette séance
https://www.youtube.com/watch?v=KsO3Uqf_oxw&t=6115s

I. Création et utilisation des annotations en java

1. Dans votre projets "Mes TPs Java", créer un package "ma.education.tp6.annotations"
2. Créer une annotation Programmer, les annotations sont créées en utilisant le mot réservé @interface . Cette annotation contient les deux signatures: int id(); et String name();

```
package ma.education.tp6.annotations;
public @interface Programmer {
	abstract int id();
	String name();
}
```

3. Cette annotation sera appliquée seulement aux classes et aux interfaces. Pour le dire, l'annotation Programmer doit être annoté par l'annotation @Target

```
@Target(ElementType.TYPE)
public @interface Programmer {
	abstract int id();
	String name();
}
```

ElementType.TYPE = Class, interface (including annotation type), or enum declaration

4. Dans ma.education.tp6.annotations, créer la classe Calculatrice, en utilisant l'annotation précédente @Programmer mentionner le programmeur qui a développé la classe Calculatrice.

```
@Programmer(id=10,name="Said ALAMI")
public class Calculatrice {
}
```

5. Créer une classe TestReflectionAnnotation pour afficher les valeurs de l'annotation @Programmer utilisées dans la classe Calculatrice

```
public class TestReflectionAnnotation {
	public static void main(String[] args) {
		Class c = Calculatrice.class;
		Programmer programmer = (Programmer)
c.getDeclaredAnnotation(Programmer.class);

System.out.println(programmer.id()+":"+programmer.name());
	}
}
```

Exécuter cette classe et remarquer que l'annotation n'existe pas au moment de l'exécution
Exception in thread "main" java.lang.NullPointerException

Une annotation est définie par sa rétention, c'est-à-dire la façon dont elle sera conservée. La rétention est définie grâce à la méta-annotation @Retention. Les différentes rétentions d'annotation possibles sont :
***SOURCE :** L'annotation est accessible durant la compilation mais n'est pas intégrée dans le fichier .class généré.*
***CLASS :** L'annotation est accessible durant la compilation, elle est intégrée dans le fichier .class généré mais elle n'est pas chargée dans la JVM à l'exécution.*
***RUNTIME :** L'annotation est accessible durant la compilation, elle est intégrée dans le fichier class généré et elle est chargée dans la JVM à l'exécution. Elle est accessible par introspection (la reflection).*

6. Ajouter alors @Retention(RetentionPolicy.RUNTIME) à l'annotation @Programmer et exécuter encore une fois la classe TestReflectionAnnotation

```
package ma.education.tp6.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface Programmer {
	abstract int id();
	String name();
}
```

7. Créer une classe fille de Calculatrice et appeler la CalculatriceMath. Est ce que la classe fille va hériter l'annotation @Programmer de sa classe mère Calculatrice.

```
public class CalculatriceMath extends Calculatrice{
}
```

8. Changer la classe TestReflectionAnnotation pour vérifier si CalcutriceMath est aussi annotée par @Programmer

```
public class TestReflectionAnnotation {
      public static void main(String[] args) {
            Class c = CalculatriceMath.class;
            Programmer programmer = (Programmer)
c.getAnnotation(Programmer.class);

System.out.println(programmer.id()+":"+programmer.name());
      }
}
```

N'oublier pas de changer c.**getDeclaredAnnotation**(Programmer.class) par c.**getAnnotation**(Programmer.class)

Exécuter cette classe et remarquer l'exception : java.lang.NullPointerException

9. Annoter l'annotation @Programmer par l'annotation @Inherited et refaire l'exécution de la classe TestReflectionAnnotation. C'est quoi votre remarque?

🤔 **Annoter le package "ma.education.tp6.annotations" par l'annotation @Programmer et afficher les valeurs de l'annotation en utilisant la réflection.**

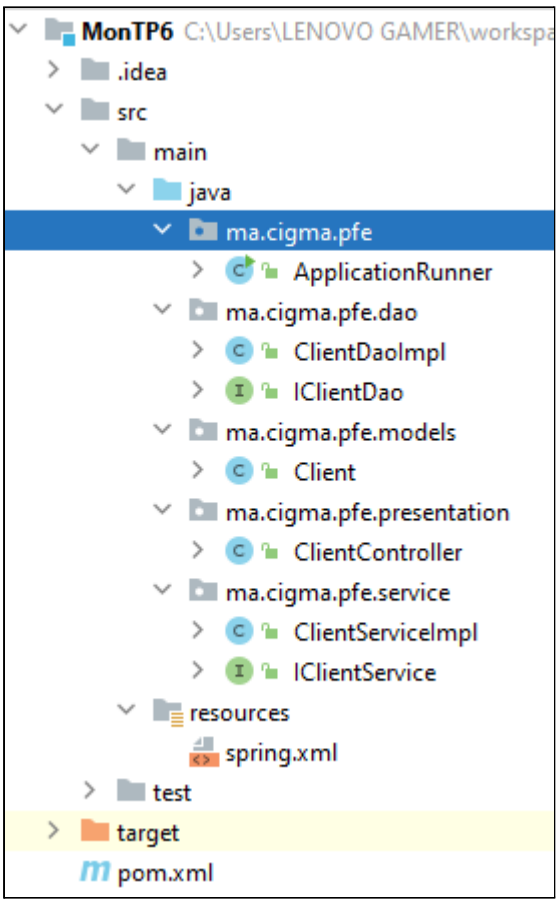
II. JPA et ses annotations

Introduction

Ce TP consiste à implémenter la couche "DAO" d'un projet [créer dans le TP6] en utilisant **l'implémentation** Hibernate de la **spécification** JPA. Il est demandé d'utiliser les annotations de base exigées par la spécification JPA.

1. Ouvrir le projet créé dans le TP6

Ci-après l'arborescence du projet du TP6



A CE STADE DE LA FORMATION, LE PROJET TP6 CONTIENDRA OBLIGATOIREMENT LES ÉLÉMENTS SUIVANTS:

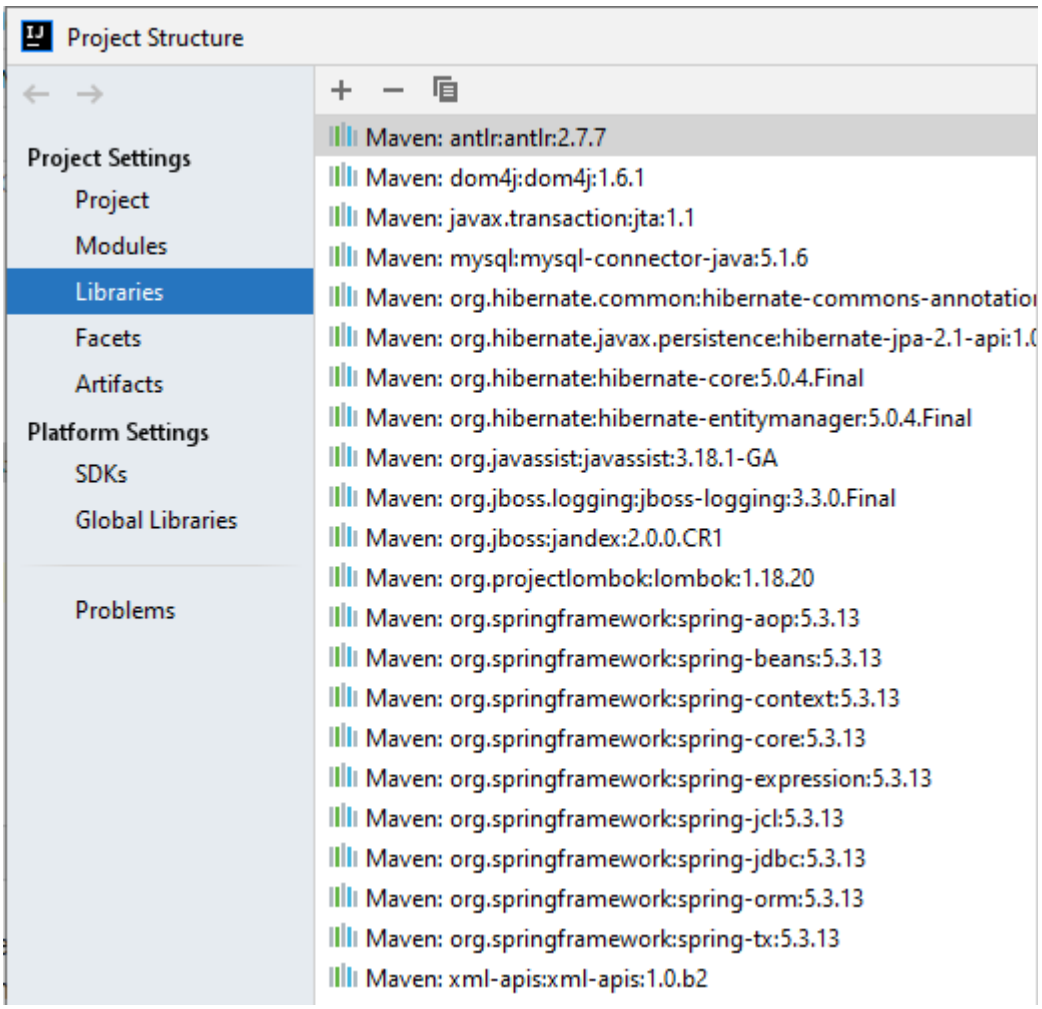
- ☐ LE FICHIER POM.XML DE MAVEN POUR L'AJOUT DES DÉPENDANCES DU PROJET
- ☐ LE FICHIER DE CONFIGURATION DU SPRING POUR L'INVERSION DU CONTRÔLE ET AUSSI L'INJECTION DE DÉPENDANCE. [DANS SRC/MAIN/RESOURCES]
- ☐ LA CLASSE MODÈLE CLIENT.JAVA [ATTRIBUTS PRIVATE @GETTER ET @SETTER DE LOMBOK]
- ☐ LA COUCHE DAO CONTIENT:
  [L'INTERFACE ICLIENTDAO ET LA CLASSE D'IMPLÉMENTATION CLIENTDAOIMPL]
- ☐ LA COUCHE SERVICE CONTIENT:
  [L'INTERFACE ICLIENTSERVICE ET LA CLASSE D'IMPLÉMENTATION CLIENTSERVICEIMPL]
- ☐ LA COUCHE PRESENTATION CONTIENT:
  [LA CLASSE CLIENTCONTROLLER]

2. AJOUTEZ AU POM.XML DE PROJET TP6 LES DÉPENDANCES SUIVANTES.

```
<properties>
    <hibernate.version>5.0.4.Final</hibernate.version>
    <spring.version>5.3.13</spring.version>
</properties>
```

```xml
<dependencies>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${spring.version}</version>
    </dependency>


    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-orm</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>${hibernate.version}</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
        <version>${hibernate.version}</version>
    </dependency>
    <dependency>
        <groupId>javax.transaction</groupId>
        <artifactId>jta</artifactId>
        <version>1.1</version>
    </dependency>

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.6</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.20</version>
        <scope>provided</scope>
    </dependency>

</dependencies>
```

VÉRIFIER QUE MAVEN A PROCÉDÉ À LA RÉCUPÉRATION DES JAR NÉCESSAIRES POUR VOTRE PROJET



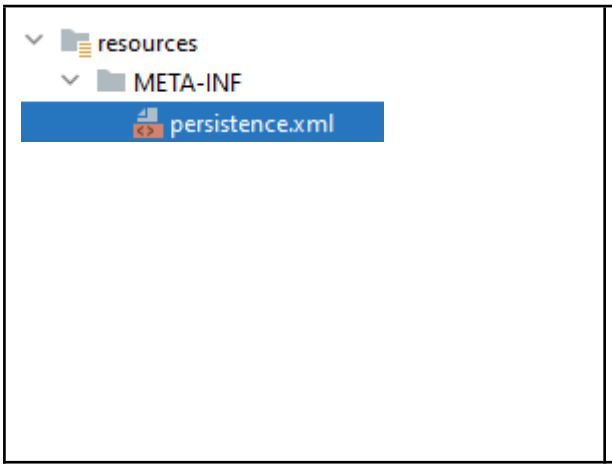
Vérifier les jars dans local repository : C:\Users\Votre_user\.m2

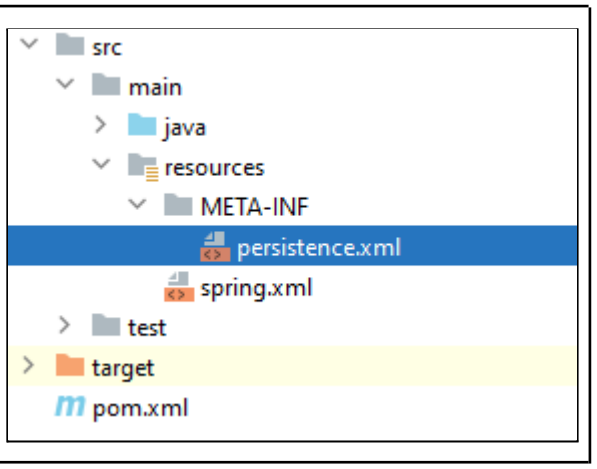
Persistence.xml : parametres du modele physique des données

Créer un "source folder" nommé "src/main/resources"

Creer dans "src/main/resources", un "folder" nomee "META-INF" en majuscule

Créer un fichier persistence.xml [en minuscule] dans src/main/resources/Meta-inf.





METTRE DANS LE FICHIER PERSISTENCE.XML LES PARAMÈTRES DE VOTRE BASE DE DONNÉES COMME SUIVANT: [IL FAUT INSTALLER LE SERVEUR MYSQL DANS VOTRE MACHINE]

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
             version="2.0">
    <persistence-unit name="unit_clients">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
        <properties>
            <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/pfe_base?createDatabaseIfNotExist=tr
ue" />
            <property name="javax.persistence.jdbc.user" value="root" />
            <property name="javax.persistence.jdbc.password" value="root"
/>
            <property name="javax.persistence.jdbc.driver"
value="com.mysql.jdbc.Driver" />
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true" />
            <property name="hibernate.hbm2ddl.auto" value="create" />
            <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQL5Dialect" />
        </properties>
    </persistence-unit>
</persistence>
```

MODIFIER LA CLASSE MODÈLE : MA.CIGMA.PFE.MODELS.CLIENT

UTILISER LA ANNOTATIONS SUIVANTES DANS LA CLASSE ENTITÉ CLIENT ET SE RAPPELER DU ROLE DE CHAQUE ANNOTATION @ENTITY @TABLE @ID @GENERATEDVALUE @COLUMN @TRANSIENT

@ENTITY	DÉFINIR UNE CLASSE MODÈLE COMME ENTITÉ À GÉRER PAR L'IMPLÉMENTATION HIBERNATE, SINON VOUS AUREZ L'EXCEPTION UNKNOWN ENTITY

@Table	Donner le nom de table équivalente à la classe entité au niveau de la base de donnée
@Id	Définir la colonne équivalent à la clé primaire dans votre table. Il aussi important de connaître @IdClass et @EmbeddedId qui seront traités dans le prochain Tp
@GeneratedValue	A utiliser si la clé doit être générée et non pas affecté dans les objets de la classe entity
@Column	À utiliser si le nom de la colonne est différent du nom de l'attribut de la classe entity.

Vous trouver ci-après la classe Client.java après modification

```
package ma.cigma.pfe.models;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.Setter;

import javax.persistence.*;

@Getter
@Setter
@AllArgsConstructor
@Entity(name = "TClients")
public class Client {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    public Client(String name) {
        this.name = name;
    }
    public Client() {
    }
    @Column
    private String name;

    @Override
    public String toString() {
        return "Client{" +
                "id=" + id +
                ", name='" + name + '\'' +
                '}';
    }
}
```

Interface de la couche DAO et son implémentation

Définir les méthodes que la couche DAO peut contenir dans l'interface ma.cigma.pfe.dao.IClientDao

Dans un premier temps une seule méthode save(Client c) sera exposée pour la partie service. Ci-après l'interface en question:

```
package ma.cigma.pfe.dao;
import ma.cigma.pfe.models.Client;
public interface IClientDao {
   boolean save(Client c);
}
```

Implémenter la méthode : save(Client b) dans une classe ma.cigma.pfe.dao.ClientDaoImpl

Utiliser un objet de type EntityManagerFactory et par la suite un objet EntityManager.

```
EntityManagerFactory emf=
Persistence.createEntityManagerFactory("unit_clients");
EntityManager em=emf.createEntityManager();
```

Utiliser une transaction dans la méthode save comme suivant:

```
@Override
public boolean save(Client c) {
   em.getTransaction().begin();
   em.persist(c);
   em.getTransaction().commit();
   return true;
}
```

La classe ma.cigma.pfe.dao.ClientDaoImpl devient:

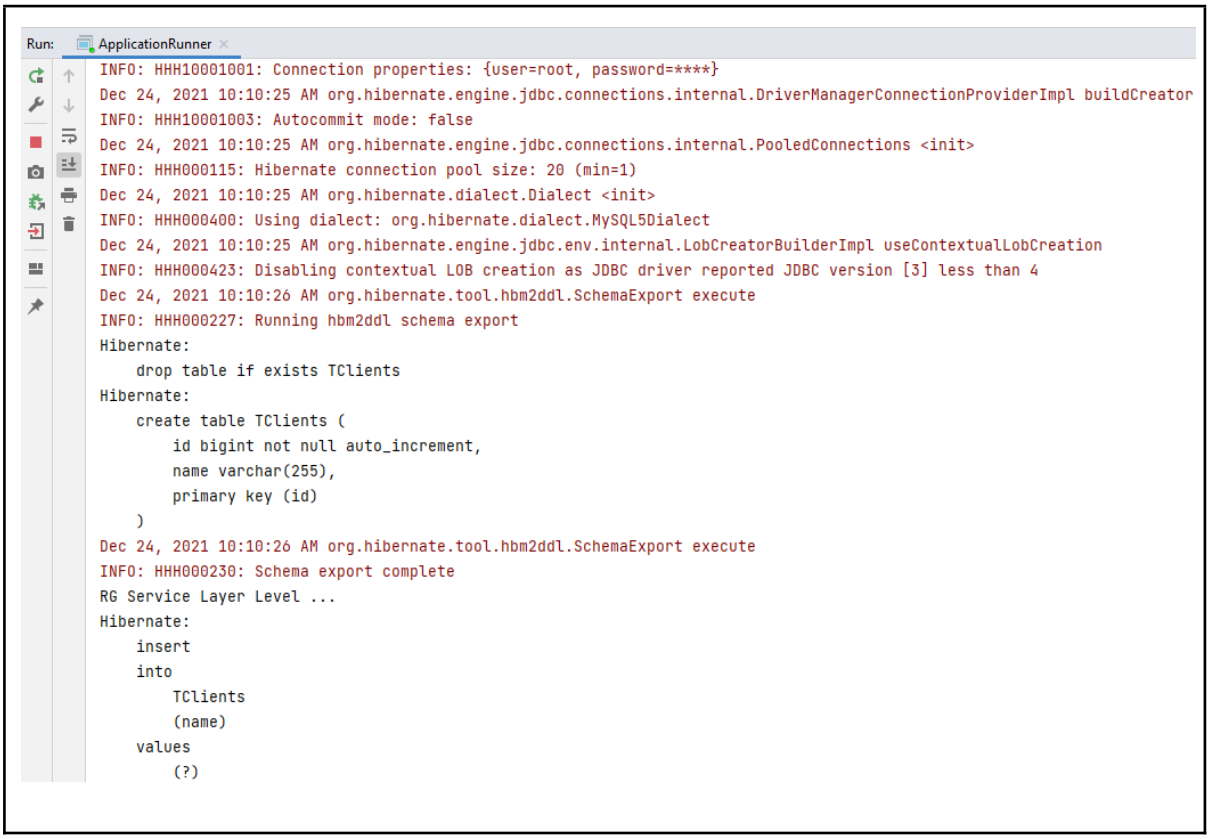
```
package ma.cigma.pfe.dao;

import ma.cigma.pfe.models.Client;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.PersistenceContext;

public class ClientDaoImpl implements  IClientDao{

   EntityManagerFactory emf=
Persistence.createEntityManagerFactory("unit_clients");
   EntityManager em=emf.createEntityManager();
```

```
    public ClientDaoImpl() {
    }
    @Override
    public boolean save(Client c) {
        em.getTransaction().begin();
        em.persist(c);
        em.getTransaction().commit();
        return true;
    }
}
```

MAINTENANT MODIFIER LA CLASSE MA.CIGMA.PFE.APPLICATIONRUNNER

```
package ma.cigma.pfe;

import ma.cigma.pfe.models.Client;
import ma.cigma.pfe.presentation.ClientController;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ApplicationRunner {

    public static void main(String[] args) {
        ApplicationContext context= new
ClassPathXmlApplicationContext("spring.xml");
        ClientController ctrl = (ClientController)
context.getBean("idCtrl");
        Client clt = new Client("OMAR");
        ctrl.save(clt);
    }
}
```

EXÉCUTER LA CLASSE MA.CIGMA.PFE.APPLICATIONRUNNER ET VERIFIER LA CONSOLE


Run: ApplicationRunner
INFO: HHH10001001: Connection properties: {user=root, password=****}
Dec 24, 2021 10:10:25 AM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001003: Autocommit mode: false
Dec 24, 2021 10:10:25 AM org.hibernate.engine.jdbc.connections.internal.PooledConnections <init>
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
Dec 24, 2021 10:10:25 AM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQL5Dialect
Dec 24, 2021 10:10:25 AM org.hibernate.engine.jdbc.env.internal.LobCreatorBuilderImpl useContextualLobCreation
INFO: HHH000423: Disabling contextual LOB creation as JDBC driver reported JDBC version [3] less than 4
Dec 24, 2021 10:10:26 AM org.hibernate.tool.hbm2ddl.SchemaExport execute
INFO: HHH000227: Running hbm2ddl schema export
Hibernate:
drop table if exists TClients
Hibernate:
create table TClients (
id bigint not null auto_increment,
name varchar(255),
primary key (id)
)
Dec 24, 2021 10:10:26 AM org.hibernate.tool.hbm2ddl.SchemaExport execute
INFO: HHH000230: Schema export complete
RG Service Layer Level ...
Hibernate:
insert
into
TClients
(name)
values
(?)