Université Sultan Moulay Slimane - Béni Mellal
École Nationale des Sciences Appliquées - Khouribga

JAVAEE

COMPTE RENDU

ATELIER 2

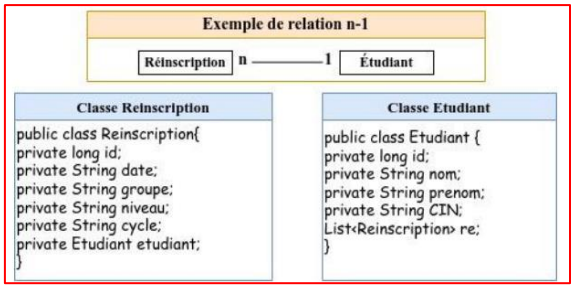
Réalisé par : *EL FATHI Zakaria*

Encadré par : *Pr. BAKKOURI Ibtissam*

SOMMAIRE

INTRODUCTION

Dans le cadre de cet atelier, nous souhaitons créer une application de gestion de l'inscription des étudiants. Pour ce faire, nous avons proposé le diagramme de classe suivant, qui contient les classes "Reinscription" et "Etudiant".



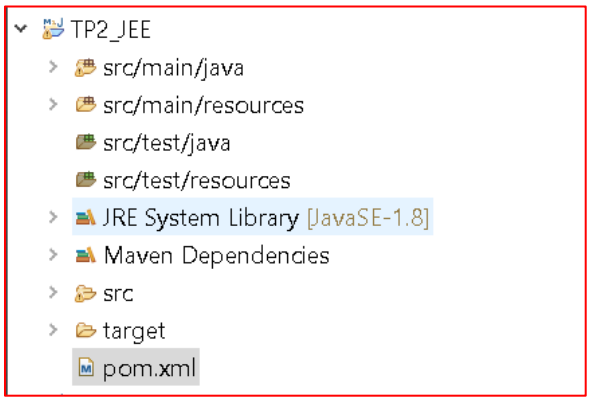
Nous allons suivre plusieurs étapes pour créer cette application. Tout d'abord, nous allons créer un projet Java et ajouter les bibliothèques nécessaires telles que Hibernate-JPA et le pilote de la base de données MySql. Ensuite, nous allons créer les entités "Etudiant" et "Reinscription" en ajoutant les annotations appropriées telles que "Entity", "Id", "GeneratedValue", "Column", "Table", etc.

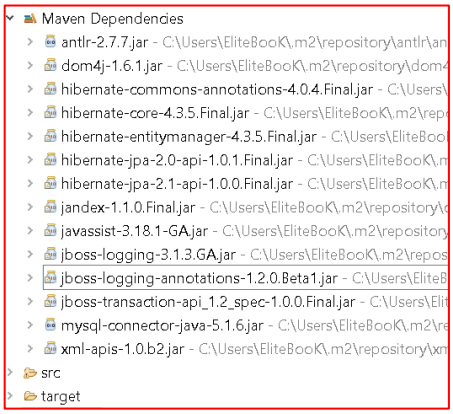
Nous allons également créer le fichier de configuration Hibernate pour permettre à notre application de se connecter à la base de données sous MySql. Nous allons ensuite créer une base de données sous MySql et y ajouter trois étudiants pour effectuer des opérations de gestion d'inscription telles que l'affichage de la liste des étudiants, la recherche des réinscriptions d'un étudiant donné, la suppression d'un étudiant, la modification des informations d'un étudiant, et l'affichage de la liste des étudiants dont le niveau est en 3ème année.

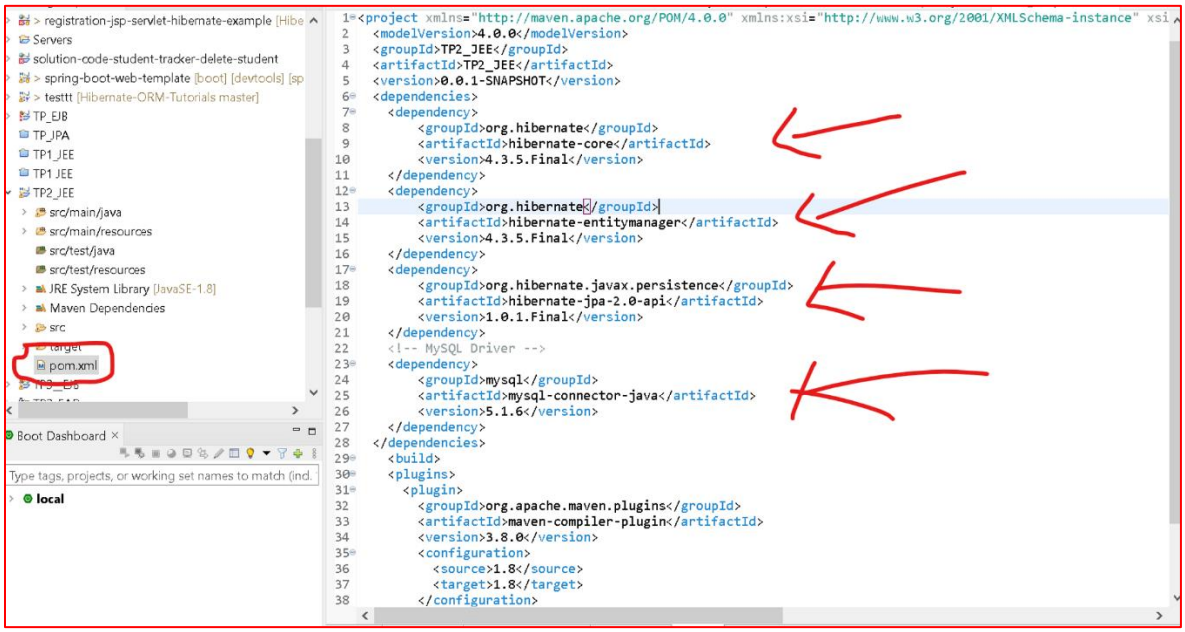
L'ensemble de ces étapes nous permettra de créer une application de gestion d'inscription des étudiants complète et fonctionnelle.

RÉALISATION:

1. *Créer un projet Java et rajouter les bibliothèques nécessaires (Hibernate-JPA et le Pilote de la base de données MySql).*







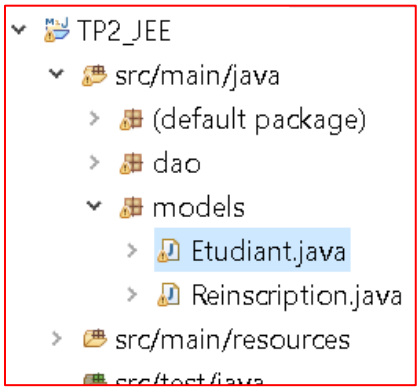
2. *Création de l'entité Etudiant et ajouter les annotations (Entity, Id, GeneratedValue, Column, Table…).*

La classe Etudiant qui représente une entité dans notre application de gestion de l'inscription des étudiants. La classe est annotée avec @Entity, ce qui indique à Hibernate qu'elle doit être persistée dans la base de données.

La classe contient des champs pour stocker les informations de l'étudiant telles que son nom, son prénom et son CIN (Carte d'identité nationale). Le champ id est annoté avec @Id et @GeneratedValue pour indiquer qu'il s'agit de la clé primaire de l'entité et que sa valeur sera générée automatiquement.

La classe contient également un champ reinscription qui est une liste d'objets de la classe Reinscription. Ce champ est annoté avec @OneToMany pour indiquer qu'il s'agit d'une relation un-à-plusieurs avec l'entité Reinscription. La propriété fetch est définie sur LAZY pour éviter de charger les réinscriptions de l'étudiant à chaque fois que l'objet Etudiant est chargé de la base de données.

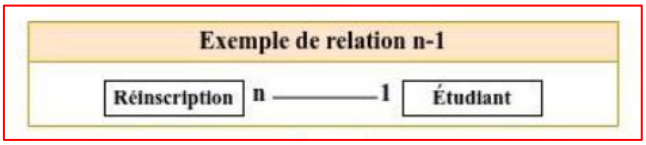
Enfin, la classe contient des getters et des setters pour accéder aux champs et modifier les valeurs de l'objet.



```java
package models;
@Entity
public class Etudiant implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String nom;
    private String prenom;
    private String cin;
    @Transient
    @OneToMany(fetch = FetchType.LAZY)
    private List<Reinscription> reinscription;

    public Etudiant() {
        super();}

    public Etudiant(String nom, String prenom, String cin) {
        super();
        this.nom = nom;
        this.prenom = prenom;
        this.cin = cin;}
    public long getId() {
        return id;}
    public void setId(long id) {
        this.id = id;}
    public String getNom() {
        return nom;}
    public void setNom(String nom) {
        this.nom = nom;}
    public String getPrenom() {
        return prenom;}
    public void setPrenom(String prenom) {
        this.prenom = prenom;}
    public String getCin() {
        return cin;}
    public void setCin(String cin) {
        this.cin = cin;}
    public List<Reinscription> getReinscription() {
        return reinscription;}
    public void setReinscription(List<Reinscription> reinscription) {
        this.reinscription = reinscription;
    }
}
```

3. *Création de l'entité Reinscription et ajouter les annotations (Entity, Id, GeneratedValue, Column, Table…)*

On ajoute cette classe 'Reinscription' donc les deux classes sont des entités JPA, annotées avec @Entity, ce qui indique qu'elles correspondent à des tables dans une base de données. La classe Etudiant a un identifiant généré automatiquement (@Id et @GeneratedValue), ainsi que des champs pour le nom, le prénom et le numéro d'identification. Elle a également une liste de reinscriptions (@OneToMany) annotée avec @Transient, ce qui signifie que cette liste n'est pas persistée dans la base de données, mais est récupérée à partir d'une autre table lors de la récupération de l'étudiant. La classe Reinscription a également un identifiant généré automatiquement, ainsi que des champs pour la date, le groupe, le niveau, le cycle et l'étudiant correspondant (@ManyToOne). La relation entre les deux classes est donc bidirectionnelle.




```
package models;

@Entity
public class Reinscription implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String date;
    private String groupe;
    private String niveau;
    private String cycle;
    @ManyToOne
    private Etudiant etudiant;


    public Reinscription() {
        super();
    }

    public Reinscription(String date, String groupe, String niveau, String cycle, Etudiant
etudiant) {
        super();
        this.date = date;
        this.groupe = groupe;
        this.niveau = niveau;
        this.cycle = cycle;
        this.etudiant = etudiant;}

    public long getId() {
        return id;}
    public void setId(long id) {
```

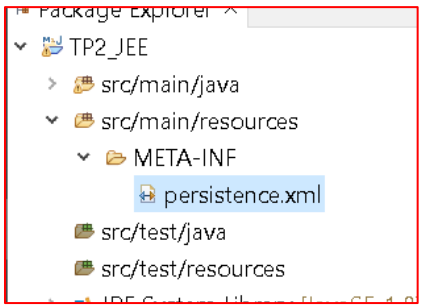
```
        this.id = id;}
    public String getDate() {
        return date;}
    public void setDate(String date) {
        this.date = date;}
    public String getGroupe() {
        return groupe;}
    public void setGroupe(String groupe) {
        this.groupe = groupe;}
    public String getNiveau() {
        return niveau;}
    public void setNiveau(String niveau) {
        this.niveau = niveau;}
    public String getCycle() {
        return cycle;}
    public void setCycle(String cycle) {
        this.cycle = cycle;}
    public Etudiant getEtudiant() {
        return etudiant;}
    public void setEtudiant(Etudiant etudiant) {
        this.etudiant = etudiant;}
}
```

4. *Création du fichier de configuration Hibernate.*

Le fichier de configuration XML pour l'utilisation de l'API JPA (Java Persistence API) avec Hibernate pour la persistance des données dans une base de données MySQL. La balise persistence définit la structure globale du fichier de configuration et contient la balise persistence-unit qui contient les propriétés de la base de données telles que le nom de l'unité de persistance, le fournisseur de persistance, les propriétés de connexion, le dialecte de la base de données et la stratégie de génération du schéma de base de données.

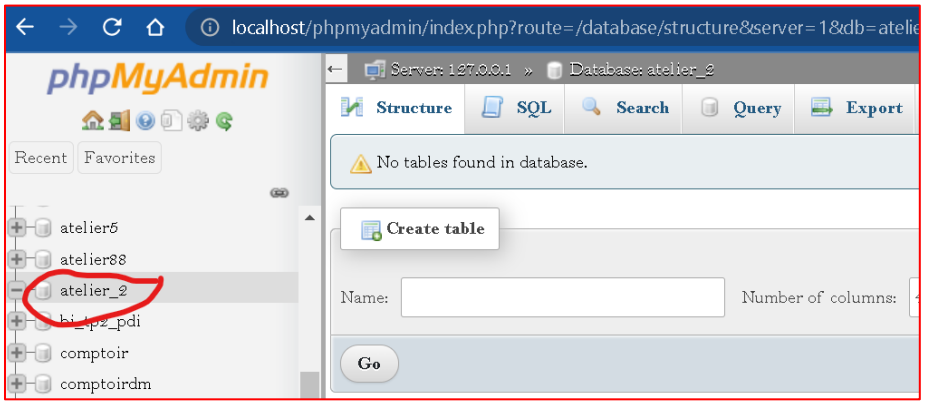
La balise provider spécifie le fournisseur de persistance qui est utilisé, et dans ce cas il s'agit de org.hibernate.jpa.HibernatePersistenceProvider. Les propriétés définies sous la balise properties sont utilisées pour configurer la connexion à la base de données MySQL, la création automatique de la base de données et l'affichage de la requête SQL générée par Hibernate.

En utilisant ce fichier de configuration, l'application Java peut être configurée pour stocker et récupérer les données dans la base de données MySQL en utilisant JPA et Hibernate.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
             http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
             version="2.1">

    <persistence-unit name="UP_CAT">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
        <properties>
            <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/atelier_2"/>
            <property name="hibernate.connection.username" value="root"/>
            <property name="hibernate.connection.password" value=""/>
            <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
            <property name="hibernate.hbm2ddl.auto" value="create"/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
        </properties>
    </persistence-unit>
</persistence>
```

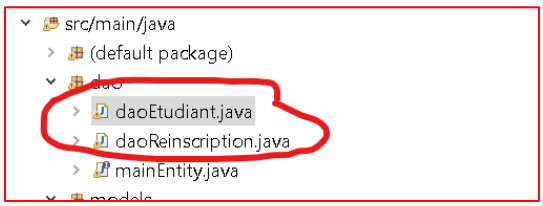
5. *Création d'une base de données sous MySql*



Dans cette étape, on va continuer par créer une interface qui contient les méthodes, des opérations CRUD demandées dans la suite de cet Atelier:

```java
package dao;
import models.Etudiant;
public interface mainEntity<entity>{
    public entity submit(entity ent);
    public List<entity> getAll();
    public entity getOneById(Long id);
    public entity update(entity ent);
    public void delete(Long id);}
```

Après on s'intéresse a la création des deux classes qui vont ensuite implémenter l'interface précédente:



daoEtudiant.java :

```java
package dao;
import models.Etudiant;
import models.Reinscription;

public class daoEtudiant implements mainEntity<Etudiant> {
    private EntityManager EM;
    public daoEtudiant(EntityManagerFactory eM) {
        super();
        EM = eM.createEntityManager();}
    public Etudiant submit(Etudiant e) {
        // TODO Auto-generated method stub
        EntityTransaction transaction = EM.getTransaction();
        transaction.begin();
        try {
            EM.persist(e);
            transaction.commit();
        }catch(Exception exc) {
            transaction.rollback();
            System.out.println("operation updateEtudiant() n'a pas pu etre aboutie!");
            exc.printStackTrace();}
        return null;
    }
    public List<Etudiant> getAll() {
        // TODO Auto-generated method stub
        EntityTransaction transaction = EM.getTransaction();
        transaction.begin();
        try {
            Query req = EM.createQuery("select p from Etudiant p");
            transaction.commit();
            return req.getResultList();
        } catch (Exception exc) {
            transaction.rollback();
            System.out.println("operation getAll_Etudiant() n'a pas pu etre aboutie!");
            exc.printStackTrace();   }
        return null; }
    public Etudiant getOneById(Long id) {
```

```
        // TODO Auto-generated method stub
        EntityTransaction transaction = EM.getTransaction();
        transaction.begin();
        try {
            Etudiant e = EM.find(Etudiant.class, id);
            transaction.commit();
            return e;
        } catch (Exception exc) {
            transaction.rollback();
            System.out.println("operation getOneById() n'a pas pu etre aboutie!");
            exc.printStackTrace(); }
        return null;
    }
    public Etudiant update(Etudiant e) {
        EntityTransaction transaction = EM.getTransaction();
        transaction.begin();
        try {
            EM.merge(e);
            // mettre à jour l'objet Etudiant dans tous les enregistrements de Reinscription
            Query query = EM.createQuery("SELECT r FROM Reinscription r WHERE r.etudiant = :etudiant");
            query.setParameter("etudiant", e);
            List<Reinscription> reinscriptions = query.getResultList();
            for (Reinscription r : reinscriptions) {
                r.setEtudiant(e);
                EM.merge(r);}
            transaction.commit();
            return e;
        } catch (Exception exc) {
            transaction.rollback();
            System.out.println("operation update() n'a pas pu etre aboutie!");
            exc.printStackTrace();  }
        return null;  }
    public void delete(Long id) {
        EntityTransaction transaction = EM.getTransaction();
        transaction.begin();
        try {
            Etudiant etudiant = EM.find(Etudiant.class, id);
            Query query = EM.createQuery("SELECT r FROM Reinscription r WHERE r.etudiant.id_etudiant = :id");
            query.setParameter("id", id);
            List<Reinscription> reinscriptions = query.getResultList();
            for (Reinscription reinscription : reinscriptions) {
                EM.remove(reinscription);
            }
            EM.remove(etudiant);
            transaction.commit();
        } catch (Exception exc) {
```

```
            transaction.rollback();
            System.out.println("Une erreur est survenue lors de la suppression de l'étudiant");
            exc.printStackTrace(); }}
    public List<Etudiant> getEtudiantsByNiveau(String niveau) {
        List<Etudiant> etudiants = new ArrayList<>();
        TypedQuery<Reinscription> query = EM.createQuery(
            "SELECT r FROM Reinscription r WHERE r.niveau = :niveau",
            Reinscription.class);
        query.setParameter("niveau", niveau);
        List<Reinscription> reinscriptions = query.getResultList();
        for (Reinscription r : reinscriptions) {
            etudiants.add(r.getEtudiant()); }
        return etudiants;  }
}
```

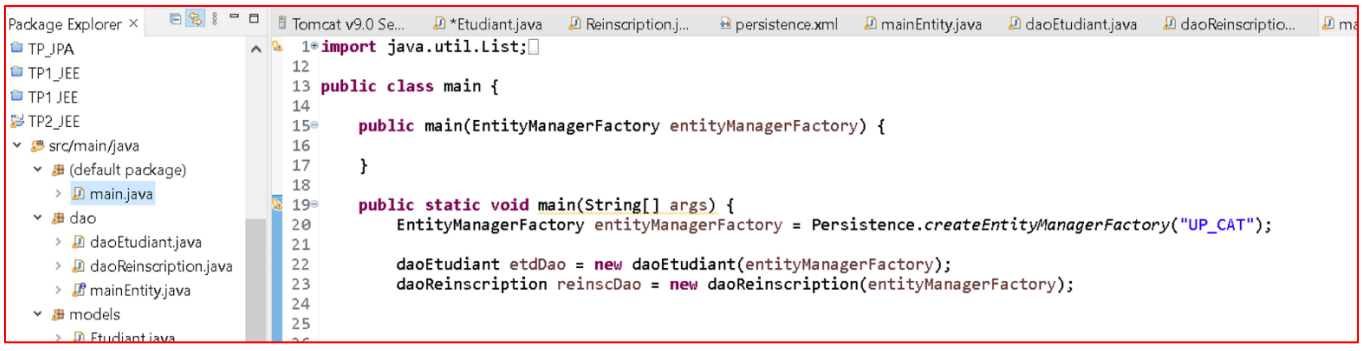
daoReinscription.java :

```
package dao;

import java.util.List;
import models.Etudiant;
import models.Reinscription;

public class daoReinscription implements mainEntity<Reinscription>{
    private EntityManager EM;
    public daoReinscription(EntityManagerFactory eM) {
        super();
        EM = eM.createEntityManager();
    }
    public Reinscription submit(Reinscription ent) {
        // TODO Auto-generated method stub
        EntityTransaction transaction = EM.getTransaction();
        transaction.begin();
        try {
            EM.persist(ent);
            transaction.commit();
        } catch (Exception exc) {
            transaction.rollback();
            System.out.println("operation submit() n'a pas pu etre aboutie!");
            exc.printStackTrace();
        }
        return null; }
    public List<Reinscription> getAll() {
        // TODO Auto-generated method stub
        EntityTransaction transaction = EM.getTransaction();
        transaction.begin();
```

```
        try {
            Query req = EM.createQuery("select r from Reinscription r");
            transaction.commit();
            return req.getResultList();

        } catch (Exception exc) {
            transaction.rollback();
            System.out.println("operation submit() n'a pas pu etre aboutie!");
            exc.printStackTrace();
        }
        return null; }
    public Reinscription getOneById(Long id) {
        // TODO Auto-generated method stub
        EntityTransaction transaction = EM.getTransaction();
        transaction.begin();
        try {
            Reinscription r = EM.find(Reinscription.class, id);
            transaction.commit();
            return r;
        } catch (Exception exc) {
            transaction.rollback();
            System.out.println("operation getOneById() n'a pas pu etre aboutie!");
            exc.printStackTrace();
        }
        return null;}
    public Reinscription update(Reinscription reinscription) {
        // TODO Auto-generated method stub
        EntityTransaction transaction = EM.getTransaction();
        transaction.begin();
        try {
            EM.merge(reinscription);
            transaction.commit();
            return reinscription;
        } catch (Exception exc) {
            transaction.rollback();
            System.out.println("operation update() n'a pas pu etre aboutie!");
            exc.printStackTrace();
        }
        return null;}
    public void delete(Long id) {
        // TODO Auto-generated method stub
        EntityTransaction Transaction = EM.getTransaction();
        Transaction.begin();
        try {
            Reinscription reinscription = EM.find(Reinscription.class, id);
            EM.remove(reinscription);
```

```
            Transaction.commit();


        } catch (Exception e) {
            Transaction.rollback();
            e.printStackTrace();
        }
    }
    public List<Reinscription> getReinscriptionByIdEtudiant(long id_etudiant) {
          EntityTransaction transaction = EM.getTransaction();
          transaction.begin();
          try {
              Query req = EM.createQuery("SELECT r FROM Reinscription r WHERE r.etudiant.id_etudiant
=:id_etudiant");
              req.setParameter("id_etudiant", id_etudiant);
              List<Reinscription> reinscriptions = req.getResultList();
              transaction.commit();
              return reinscriptions;
          } catch (Exception exc) {
              transaction.rollback();
              System.out.println("operation getReinscriptionByEtudiantId() n'a pas pu être aboutie!");
              exc.printStackTrace();
          }
          return null;
        }
}
```

6. *pour réaliser les question d'apres on aura besoin de créer une classe principale qui va utiliser les méthodes déja définies dans les classes précédentes:*



on fait une insertion de trois étudiants et trois inscriptions, comme suit:

```
import dao.daoEtudiant;
import dao.daoReinscription;
import models.Etudiant;
import models.Reinscription;
```

```java
public class main {

    public main(EntityManagerFactory entityManagerFactory) {    }

    public static void main(String[] args) {
        EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("UP_CAT");

        daoEtudiant etdDao = new daoEtudiant(entityManagerFactory);
        daoReinscription reinscDao = new daoReinscription(entityManagerFactory);

        //QUESTION 6
        Etudiant e1 = new Etudiant(44L,"EL FATHI", "ZAKARIA", "Z111111");
        Etudiant e2 = new Etudiant(60L,"ZAHIDI", "LAMYAE", "Q111111");
        Etudiant e3 = new Etudiant(88L,"OUADRASSI", "ZIAD", "ZT14555");
        etdDao.submit(e1);
        etdDao.submit(e2);
        etdDao.submit(e3);


        Reinscription r1 = new Reinscription("06/11/2022", "S4", "7", "M6", e1);
        Reinscription r2 = new Reinscription("06/11/2021", "S3", "2", "M5", e1);
        Reinscription r3 = new Reinscription("06/11/2019", "S1", "3", "M3", e1);

        reinscDao.submit(r1);
        reinscDao.submit(r2);
        reinscDao.submit(r3);

    }}
```

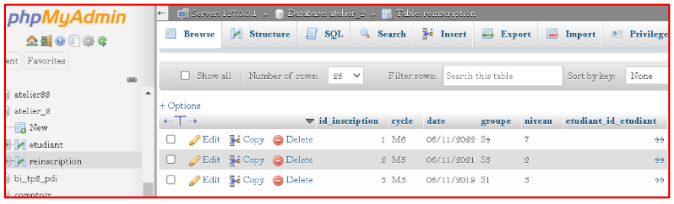


Table reinscriptions

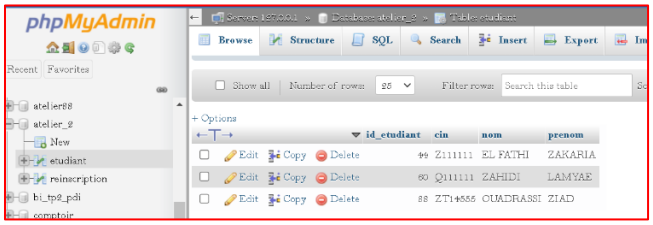


Table Etudiant

7. Affichage de la liste des étudiants.

pour ce faire on utilise(dans la classe principale) :

```
//Question 7: Recuperation de la liste des etudiants
System.out.println("____________________\n Liste des Etudiants:\n");
List<Etudiant> etudiants = etdDao.getAll();
for(Etudiant e : etudiants) {
    System.out.println("Id: " + e.getId() + ", Nom: " + e.getNom() + ", Prenom: " + e.getPrenom() + ",
CIN: " + e.getCin());
}
System.out.println("____________________");
entityManagerFactory.close();
}
```

qui utilise la fonction getAll definie dans l'implementaion de l'Etudiant comme suit:

```
public List<Etudiant> getAll() {
    // TODO Auto-generated method stub
    EntityTransaction transaction = EM.getTransaction();
    transaction.begin();
    try {
        Query req = EM.createQuery("select p from Etudiant p");
        transaction.commit();
        return req.getResultList();
    } catch (Exception exc) {
        transaction.rollback();
        System.out.println("operation getAll_Etudiant() n'a pas pu etre aboutie!");
        exc.printStackTrace();
    }
    return null;
}
```

on aura dans le console:

```
Hibernate: insert into Reinscription (cycle, date, etudiant_id_etudiant, groupe, ni
Hibernate: insert into Reinscription (cycle, date, etudiant_id_etudiant, groupe, ni

____________________
 Liste des Etudiants:

Hibernate: select etudiant0_.id_etudiant as id_etudi1_0_, etudiant0_.cin as cin2_0_
Id: 44, Nom: EL FATHI, Prenom: ZAKARIA, CIN: Z111111
Id: 60, Nom: ZAHIDI, Prenom: LAMYAE, CIN: Q111111
Id: 88, Nom: OUADRASSI, Prenom: ZIAD, CIN: ZT14555

____________________
avr. 08, 2023 4:46:44 PM org.hibernate.engine.jdbc.connections.internal.DriverManag
INFO: HHH000030: Cleaning up connection pool [jdbc:mysql://localhost:3306/atelier_2
```

8. *Affichage des réinscriptions de l'étudiant dont id = 44.*

pour ce faire on definie une fonction dans "daoReinscription.java" telle que:

```
public List<Reinscription> getReinscriptionByIdEtudiant(long id_etudiant) {
    EntityTransaction transaction = EM.getTransaction();
    transaction.begin();
    try {
        Query req = EM.createQuery("SELECT r FROM Reinscription r WHERE r.etudiant.id_etudiant
=:id_etudiant");
        req.setParameter("id_etudiant", id_etudiant);
        List<Reinscription> reinscriptions = req.getResultList();
        transaction.commit();
        return reinscriptions;
    } catch (Exception exc) {
        transaction.rollback();
        System.out.println("operation getReinscriptionByEtudiantId() n'a pas pu être aboutie!");
        exc.printStackTrace();
    }
    return null;
 }
```

et on definie la methode toString() dans le bean pour mieux afficher les details des inscriptions:

```
@Override
    public String toString() {
        return "Reinscription [id_inscription=" + id_inscription + ", date=" + date + ", groupe=" +
groupe + ", niveau="
                + niveau + ", cycle=" + cycle + "]";
    }
```

en suite on fait appel dans la classe principale:

```
//Question 8
Etudiant etud = etdDao.getOneById(44L);
List<Reinscription> reinscriptions = reinscDao.getReinscriptionByIdEtudiant(etud.getId());
if (reinscriptions != null && !reinscriptions.isEmpty()) {
    System.out.println("Reinscriptions de l'étudiant avec l'ID " + etud.getId() + ":");
    for (Reinscription reinscription : reinscriptions) {
        System.out.println(reinscription.toString());
    }
} else {
    System.out.println("Aucune reinscription trouvée pour l'étudiant avec l'ID " + etud.getId());
}
```

on aura l'affichage suivant:

```
Hibernate: select reinscript0_.id_inscription as id_inscr1_1_, reinscript0_.cycle as cycle2_1_, reinscript0_.date as d
Reinscriptions de l'étudiant avec l'ID 44:
Reinscription [id_inscription=1, date=06/11/2022, groupe=S4, niveau=7, cycle=M6]
Reinscription [id_inscription=2, date=06/11/2021, groupe=S3, niveau=2, cycle=M5]
Reinscription [id_inscription=3, date=06/11/2019, groupe=S1, niveau=3, cycle=M3]
avr. 08, 2023 11:38:43 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop
INFO: HHH000030: Cleaning up connection pool [jdbc:mysql://localhost:3306/atelier_2]
```

c'est exactement les memes inscriptions fait par l'etudiant d'ID =44 comme dans la table reinscription suivante:

+ Options

	id_inscription	cycle	date	groupe	niveau	etudiant_id_etudiant
Edit Copy Delete	1	M6	06/11/2022	S4	7	44
Edit Copy Delete	2	M5	06/11/2021	S3	2	44
Edit Copy Delete	3	M3	06/11/2019	S1	3	44
Edit Copy Delete	4	M8	06/11/2021	S5	6	60

9. *Suppression de l'étudiant dont id = 60.*

on definie une methode de suppression dans la daoEtudiant, cette méthode recherche également toutes les réinscriptions associées à l'étudiant dont l'ID est donné et les supprime, puis supprime l'étudiant lui-même. Si une erreur se produit, elle effectue un rollback de la transaction.

comme suit:

```
public void delete(Long id) {
    EntityTransaction transaction = EM.getTransaction();
    transaction.begin();
    try {
        Etudiant etudiant = EM.find(Etudiant.class, id);
        Query query = EM.createQuery("SELECT r FROM Reinscription r WHERE
r.etudiant.id_etudiant = :id");
        query.setParameter("id", id);
        List<Reinscription> reinscriptions = query.getResultList();
        for (Reinscription reinscription : reinscriptions) {
            EM.remove(reinscription);
        }
        EM.remove(etudiant);
        transaction.commit();
    } catch (Exception exc) {
        transaction.rollback();
        System.out.println("erreur de suppression de l'étudiant");
        exc.printStackTrace();
    }
}
```

dans la classe principale on aura le code qui fait appel a cette methode comme suit:

```
//QUESTION 9
System.out.println("_____");
Etudiant etud60 = etdDao.getOneById(60L);
etdDao.delete(etud60.getId());
System.out.println("methode de suppression terminee!");
```

apres execution on aura :

```
_____
Hibernate: select reinscript0_.id_inscription as id_inscr1_1_, reinscript0_.cycle as cycle2_1_, reinscript0_.d
Hibernate: delete from Reinscription where id_inscription=?
Hibernate: delete from Etudiant where id_etudiant=?
methode de suppression terminee!
avr. 08, 2023 11:57:06 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl s
INFO: HHH000030: Cleaning up connection pool [jdbc:mysql://localhost:3306/atelier_2]
```

10. Modification des informations d'étudiant dont id = 88.

pour ce faire, dans la table Reinscription,on fait parcourir tous les enregistrements de Reinscription qui ont l'ancien objet Etudiant et mettre à jour la propriété Etudiant avec le nouvel objet Etudiant. nous allons modifier la méthode update() comme suit :

```
public Etudiant update(Etudiant e) {
    EntityTransaction transaction = EM.getTransaction();
    transaction.begin();
    try {
        EM.merge(e);

        // mettre à jour l'objet Etudiant dans tous les enregistrements de Reinscription
        Query query = EM.createQuery("SELECT r FROM Reinscription r WHERE r.etudiant = :etudiant");
        query.setParameter("etudiant", e);
        List<Reinscription> reinscriptions = query.getResultList();
        for (Reinscription r : reinscriptions) {
            r.setEtudiant(e);
            EM.merge(r);
        }
        transaction.commit();
        return e;
    } catch (Exception exc) {
        transaction.rollback();
        System.out.println("operation update() n'a pas pu etre aboutie!");
        exc.printStackTrace();
    }
    return null;
}
```

on fait appel dans la classe principale:

```
//QUESTION 10
System.out.println("_____");
Etudiant etud88 = etdDao.getOneById(88L);
etud88.setCin("C44444");
etdDao.update(etud88);
System.out.println("methode de modification terminee!");
```

donc on aura dans la table :

	id_etudiant	cin	nom	prenom
Edit Copy Delete	44	Z111111	EL FATHI	ZAKARIA
Edit Copy Delete	56	W516668	TALIB	ABDO
Edit Copy Delete	88	C44444	OUADRASSI	ZIAD

Check all With selected: Edit Copy Delete Ex

11. Affichage de la liste des étudiants dont le niveau est en 3ème année

Pour ce faire, on definie la méthode "getEtudiantsByNiveau" qui permet de récupérer une liste d'étudiants dont le niveau est passé en paramètre. Elle utilise une requête JPQL pour sélectionner les réinscriptions dont le niveau correspond à celui passé en paramètre. Ensuite, elle récupère les étudiants associés à ces réinscriptions et les ajoute à une liste qu'elle renvoie.

```
public List<Etudiant> getEtudiantsByNiveau(String niveau) {
    List<Etudiant> etudiants = new ArrayList<>();
    TypedQuery<Reinscription> query = EM.createQuery(
        "SELECT r FROM Reinscription r WHERE r.niveau = :niveau",
        Reinscription.class);
    query.setParameter("niveau", niveau);
    List<Reinscription> reinscriptions = query.getResultList();
    for (Reinscription r : reinscriptions) {
        etudiants.add(r.getEtudiant());
    }
    return etudiants;
}
```

dans la classe principale on aura:

```
//QUESTION 11
System.out.println("_____");
List<Etudiant> etudiantsNiveau3 = etdDao.getEtudiantsByNiveau("3");
System.out.println("Liste des étudiants en 3ème année :");
for (Etudiant etudiant : etudiantsNiveau3) {
    System.out.println(etudiant.toString());
}
System.out.println("methode terminee!");
```

donc on recupere les etudiants en 3eme annee comme suit:

```
Hibernate: select reinscript0_.id_inscription as id_inscr1_1_, reinscript0_.cycle as cycle2_1_,
Liste des étudiants en 3ème année :
Etudiant [id_etudiant=44, nom=EL FATHI, prenom=ZAKARIA, cin=Z111111]
Etudiant [id_etudiant=56, nom=TALIB, prenom=ABDO, cin=W516668]
methode terminee!
avr. 09, 2023 12:55:01 AM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectio
INFO: HHH000030: Cleaning up connection pool [jdbc:mysql://localhost:3306/atelier_2]
```

Dans la suite on obient la classe principale 'Main' comme suit:

```
import javax.persistence.*;
import dao.daoEtudiant;
import dao.daoReinscription;
import models.Etudiant;
import models.Reinscription;

public class main {

    public main(EntityManagerFactory entityManagerFactory) {}

    public static void main(String[] args) {
        EntityManagerFactory entityManagerFactory =
Persistence.createEntityManagerFactory("UP_CAT");

        daoEtudiant etdDao = new daoEtudiant(entityManagerFactory);
        daoReinscription reinscDao = new daoReinscription(entityManagerFactory);

        //QUESTION 6
        Etudiant e1 = new Etudiant(44L,"EL FATHI", "ZAKARIA", "Z111111");
        Etudiant e2 = new Etudiant(60L,"ZAHIDI", "LAMYAE", "Q111111");
        Etudiant e3 = new Etudiant(88L,"OUADRASSI", "ZIAD", "ZT14555");
        Etudiant e4 = new Etudiant(56L,"TALIB", "ABDO", "W516668");
        etdDao.submit(e1);
        etdDao.submit(e2);
        etdDao.submit(e3);
        etdDao.submit(e4);


        Reinscription r1 = new Reinscription("06/11/2022", "S4", "7", "M6", e1);
        Reinscription r2 = new Reinscription("06/11/2021", "S3", "2", "M5", e1);
        Reinscription r3 = new Reinscription("06/11/2019", "S1", "3", "M3", e1);
        Reinscription r4 = new Reinscription("06/11/2021", "S5", "4", "M8", e2);
        Reinscription r5 = new Reinscription("06/11/2021", "S3", "3", "M9", e4);
        reinscDao.submit(r1);
        reinscDao.submit(r2);
        reinscDao.submit(r3);
        reinscDao.submit(r4);
```

```
        reinscDao.submit(r5);


        //Question 7: Recuperation de la liste des etudiants
        System.out.println("____________________\n Liste des Etudiants:\n");
        List<Etudiant> etudiants = etdDao.getAll();
        for(Etudiant e : etudiants) {
            System.out.println("Id: " + e.getId() + ", Nom: " + e.getNom() + ", Prenom: " +
e.getPrenom() + ", CIN: " + e.getCin());
        }
        System.out.println("____________________");


        //Question 8
        Etudiant etud = etdDao.getOneById(44L);
        List<Reinscription> reinscriptions =
reinscDao.getReinscriptionByIdEtudiant(etud.getId());
        if (reinscriptions != null && !reinscriptions.isEmpty()) {
            System.out.println("Reinscriptions de l'étudiant avec l'ID " + etud.getId() +
":");
            for (Reinscription reinscription : reinscriptions) {
                System.out.println(reinscription.toString());
            }
        } else {
            System.out.println("Aucune reinscription trouvée pour l'étudiant avec l'ID " +
etud.getId());
        }


        //QUESTION 9
        System.out.println("_______________________________");
        Etudiant etud60 = etdDao.getOneById(60L);
        etdDao.delete(etud60.getId());
        System.out.println("methode de suppression terminee!");


        //QUESTION 10
        System.out.println("_______________________________");
        Etudiant etud88 = etdDao.getOneById(88L);
        etud88.setCin("C44444");
        etdDao.update(etud88);
        System.out.println("methode de modification terminee!");


        //QUESTION 11
        System.out.println("_______________________________");
```

```
        List<Etudiant> etudiantsNiveau3 = etdDao.getEtudiantsByNiveau("3");
        System.out.println("Liste des étudiants en 3ème année :");
        for (Etudiant etudiant : etudiantsNiveau3) {
            System.out.println(etudiant.toString());
        }
        System.out.println("methode terminee!");

        entityManagerFactory.close();
    }

}
```

CONCLUSION:

En conclusion, nous avons présenté une proposition pour la création d'une application de gestion d'inscription des étudiants. En utilisant le diagramme de classe, nous avons détaillé les différentes étapes nécessaires pour la mise en place du projet, y compris la création des entités Etudiant et Reinscription avec les annotations appropriées, la configuration de Hibernate, et la création d'une base de données sous MySql. Nous avons également présenté une série de fonctionnalités telles que l'affichage de la liste des étudiants, l'affichage des réinscriptions pour un étudiant spécifique, la suppression et la modification d'étudiants, ainsi que l'affichage de la liste des étudiants en fonction de leur niveau. Cette application de gestion de l'inscription des étudiants peut être étendue en ajoutant de nouvelles fonctionnalités et en adaptant le modèle de données pour répondre à d'autres besoins spécifiques.