**4 Лабораторна робота № 4.  
Вивчення об’єктно-реляційного відображення**

**4.1 Мета роботи**

В результаті виконання лабораторної роботи студент має отримати такі знання і навички:

– ознайомитися з технологією об’єктно-реляційного відображення;

– навчитися працювати з базою даних на рівні об’єктів та створювати запити до бази, використовуючи мову запитів JPQL;

– отримати навички створення програми для роботи з реляційною базою даних через об’єктно-реляційне відображення.

**4.2 Результат виконання роботи**

**4.2.1 Файл persistence.xml**

Текст файлу persistence.xml зображено в лістингу 1.

Лістинг 1 – Файл persistence.xml

<?xml version="1.0" encoding="UTF-8"?>

<persistence version="2.1"

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">

<persistence-unit name="jpaOlenchenko" transaction-type="RESOURCE\_LOCAL">

<class>jpaOlenchenko.Audience</class>

<class>jpaOlenchenko.Corps</class>

<class>jpaOlenchenko.Furniture</class>

<properties>

<property name="javax.persistence.jdbc.url" value="jdbc:derby:dbOlenchenko;create=true;upgrade=true"/>

<property name="javax.persistence.jdbc.user" value="Olenchenko"/>

<property name="javax.persistence.jdbc.password" value="1212"/>

<property name="javax.persistence.jdbc.driver" value="org.apache.derby.jdbc.EmbeddedDriver"/>

</properties>

</persistence-unit>

</persistence>

**4.2.2 Робота з базою даних**

4.2.2.1 Використання іменованого запиту з класу моделі

Текст класу TestNamedQuery в пакеті test зображено в лістингу 2, а результат на рисунку 1.

Лістинг 2 – Клас TestNamedQuery

**public** **class** TestNamedQuery {

**public** **static** **void** main(String[] args) {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpaOlenchenko");

EntityManager em = emf.createEntityManager();

Query q = em.createNamedQuery("Corps.findAll");

@SuppressWarnings("unchecked")

List<Corps> list = q.getResultList();

**for** (Corps cth : list) {

System.***out***.println(cth.getId() + " " + cth.getAudience() + " " + cth.getCorps() + " " + cth.getFurniture());

**for**(Audience gr : cth.getAudiences()) {

System.***out***.println("\t"+gr.getId() + " " + gr.getArea() + " "+ gr.getFloor() + " "+ gr.getNumber());}}}}

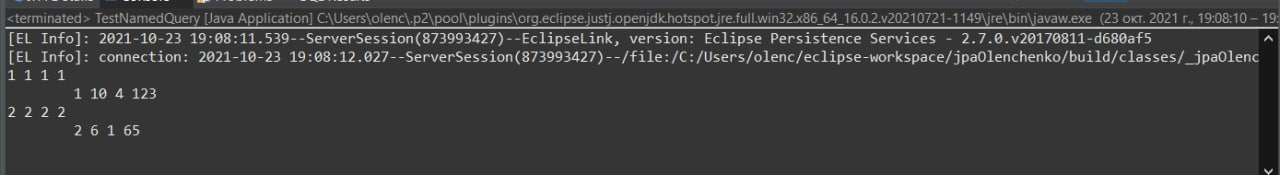


Рисунок 1 – Результат TestNamedQuery

4.2.2.2 Використання запиту з параметром

Текст класу TestParamemterQuery зображено в лістингу 3, а результат на рисунку 2.

Лістинг 3 – Клас TestParamemterQuery

**public** **class** TestParamemterQuery {

**public** **static** **void** main(String[] args) {

EntityManagerFactory emf =

Persistence.createEntityManagerFactory("jpaOlenchenko");

EntityManager em = emf.createEntityManager();

String sq = "select gr from Audience gr where gr.area = ?1";

Query q = em.createQuery(sq);

q.setParameter(1, "10");

@SuppressWarnings("unchecked")

List<Audience> list = q.getResultList();

**for** (Audience gr : list) {

System.***out***.println("\t"+gr.getId() + " " + gr.getArea() + " "+ gr.getFloor() + " "+ gr.getNumber());}}}

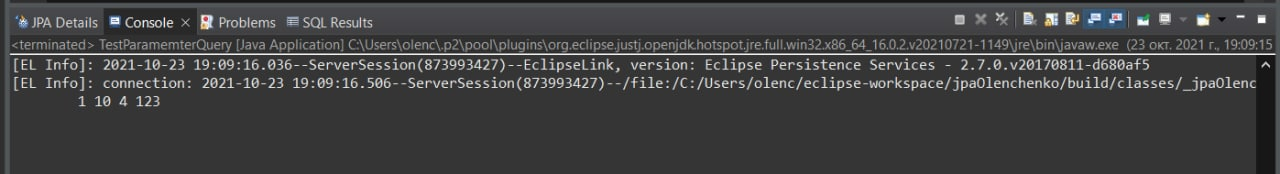


Рисунок 2 – Результат TestParamemterQuery

4.2.2.3 Оновлення інформації у базі даних

Текст класу TestUpdate зображено в лістингу 4, а результат на рисунку 3.

Лістинг 4 – Клас TestUpdate

**public** **class** TestUpdate {

**public** **static** **void** main(String[] args) {

EntityManagerFactory emf =

Persistence.createEntityManagerFactory("jpaOlenchenko");

EntityManager em = emf.createEntityManager();

Query q = em.createNamedQuery("Audience.findAll");

//Output graduates list before update

List<Audience> list = q.getResultList();

**for** (Audience gr : list) System.***out***.println("\t"+gr.getId() + " " + gr.getArea() + " "+ gr.getFloor() + " "+ gr.getNumber());

//Update floor name with id = 1

em.getTransaction().begin();

Audience grd = em.find(Audience.**class**, 1);

System.***out***.println("\nПісля\n");

grd.setFloor("1212");

em.getTransaction().commit();

//Output Audience list after update

list = q.getResultList();

**for** (Audience gr : list) System.***out***.println("\t"+gr.getId() + " " + gr.getArea() + " "+ gr.getFloor() + " "+ gr.getNumber());

}}

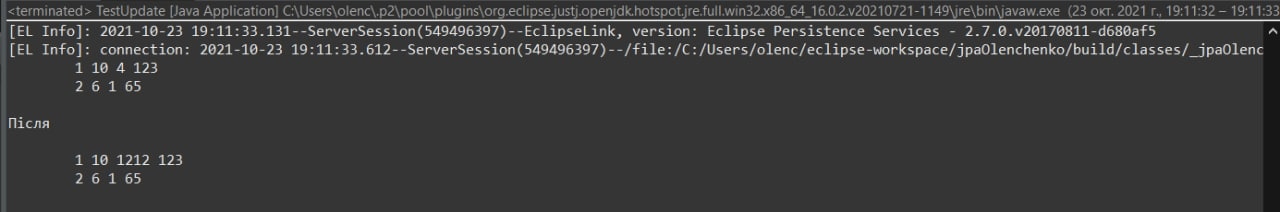


Рисунок 3 – Результат TestUpdate

4.2.2.4 Створення нового запису у таблиці

Текст класу TestAddCorps зображено в лістингу 5, а результат на рисунку 4.

Лістинг 5 – Клас TestAddCorps

**public** **class** TestAddCorps {

**public** **static** **void** main(String[] args) {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpaOlenchenko");

EntityManager em = emf.createEntityManager();

Query q = em.createNamedQuery("Corps.findAll");

@SuppressWarnings("unchecked")

List<Corps> list = q.getResultList();

**for** (Corps cth : list) {

System.***out***.println(cth.getId() + " " + cth.getAudience() + " " + cth.getCorps() + " " + cth.getFurniture());

**for**(Audience gr : cth.getAudiences()) {

System.***out***.println("\t"+gr.getId() + " " + gr.getArea() + " "+ gr.getFloor() + " "+ gr.getNumber());}}

// Транзакція створення нової кафедри

Corps cth = **new** Corps();

cth.setAudience("91");

cth.setCorps("91");

cth.setFurniture("991");

em.getTransaction().begin();

em.persist(cth);

em.getTransaction().commit();

//

System.***out***.println("\n->\n");

@SuppressWarnings("unchecked")

List<Corps> list1 = q.getResultList();

**for** (Corps cth1 : list1) {

System.***out***.println(cth1.getId() + " " + cth1.getAudience() + " " + cth1.getCorps() + " " + cth1.getFurniture());

**for**(Audience gr : cth1.getAudiences()) {

System.***out***.println("\t"+gr.getId() + " " + gr.getArea() + " "+ gr.getFloor() + " "+ gr.getNumber());}}}}

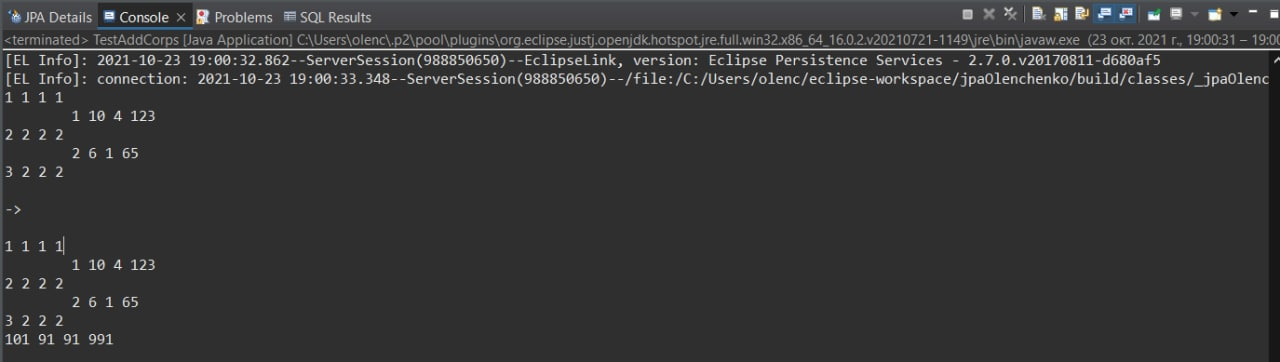


Рисунок 4 – Результат TestAddCorps

4.2.2.5 Вилучення запису із таблиці по id

Текст класу TestDelById зображено в лістингу 6, а результат на рисунку 5.

Лістинг 6 – Клас TestDelById

**public** **class** TestDelById {

**public** **static** **void** main(String[] args) {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpaOlenchenko");

EntityManager em = emf.createEntityManager();

Query q = em.createNamedQuery("Corps.findAll");

@SuppressWarnings("unchecked")

List<Corps> list = q.getResultList();

**for** (Corps cth : list) {

System.***out***.println(cth.getId() + " " + cth.getAudience() + " " + cth.getCorps() + " " + cth.getFurniture());

**for**(Audience gr : cth.getAudiences()) {

System.***out***.println("\t"+gr.getId() + " " + gr.getArea() + " "+ gr.getFloor() + " "+ gr.getNumber());

}}

em.getTransaction().begin();

Corps st = em.find(Corps.**class**, 101);

em.remove(st);

em.getTransaction().commit();

System.***out***.println("\nПісля\n");

@SuppressWarnings("unchecked")

List<Corps> list1 = q.getResultList();

**for** (Corps cth : list1) {

System.***out***.println(cth.getId() + " " + cth.getAudience() + " " + cth.getCorps() + " " + cth.getFurniture());

**for**(Audience gr : cth.getAudiences()) {

System.***out***.println("\t"+gr.getId() + " " + gr.getArea() + " "+ gr.getFloor() + " "+ gr.getNumber()); } }}}

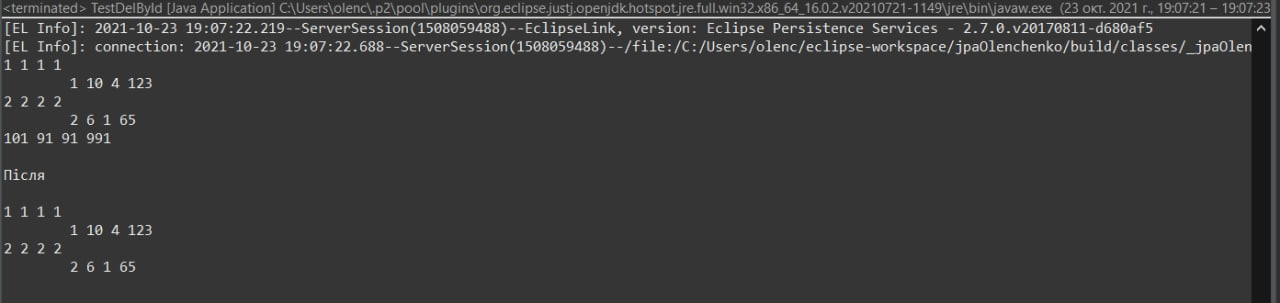


Рисунок 5 – Результат TestDelById

4.2.2.6 Вилучення запису із таблиці з використанням JPQL

Текст класу TestDelByName зображено в лістингу 7, результат на рисунку 6.

Лістинг 7 – Клас TestDelByName

**public** **class** TestDelByName {

**public** **static** **void** main(String[] args) {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpaOlenchenko");

EntityManager em = emf.createEntityManager();

Query q = em.createNamedQuery("Corps.findAll");

@SuppressWarnings("unchecked")

List<Corps> list = q.getResultList();

**for** (Corps cth : list) {

System.***out***.println(cth.getId() + " " + cth.getAudience() + " " + cth.getCorps() + " " + cth.getFurniture());

**for**(Audience gr : cth.getAudiences()) {

System.***out***.println("\t"+gr.getId() + " " + gr.getArea() + " "+ gr.getFloor() + " "+ gr.getNumber());

}}

String sq = "DELETE FROM Corps cth WHERE cth.corps = ?1";

Query qDel = em.createQuery(sq);

qDel.setParameter(1, "1");

em.getTransaction().begin();

qDel.executeUpdate();

em.getTransaction().commit();

@SuppressWarnings("unchecked")

List<Corps> list1 = q.getResultList();

**for** (Corps cth1 : list1) {

System.***out***.println(cth1.getId() + " " + cth1.getAudience() + " " + cth1.getCorps() + " " + cth1.getFurniture());

**for**(Audience gr : cth1.getAudiences()) {

System.***out***.println("\t"+gr.getId() + " " + gr.getArea() + " "+ gr.getFloor() + " "+ gr.getNumber());}}}}

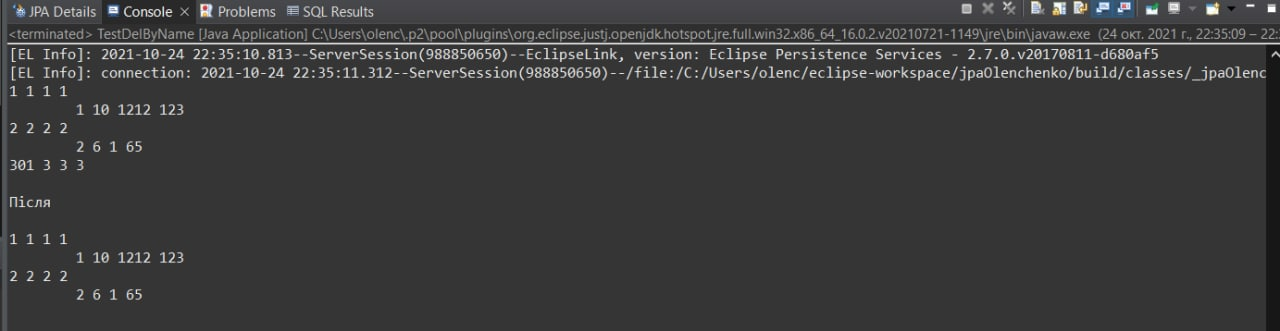


Рисунок 6 – Результат TestDelByName

4.2.2.7 Використання запиту на SQL

Текст класу TestSql зображено в лістингу 8, а результат на рисунку 7.

Лістинг 8 – Клас TestSql

**public** **class** TestSql {

**public** **static** **void** main(String[] args) {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpaOlenchenko");

EntityManager em = emf.createEntityManager();

// SQL query

String sql = "SELECT \* FROM Audience WHERE Audience.number >= '0'";

Query qSql = em.createNativeQuery(sql);

em.getTransaction().begin();

@SuppressWarnings("unchecked")

List<Object[]> list = qSql.getResultList();

em.getTransaction().commit();

// Output list

**for** (Object[] arr : list) {

**for** (**int** i = 0; i < arr.length; i++) {

System.***out***.print(arr[i] + "\t");

}

System.***out***.println();

}

}

}

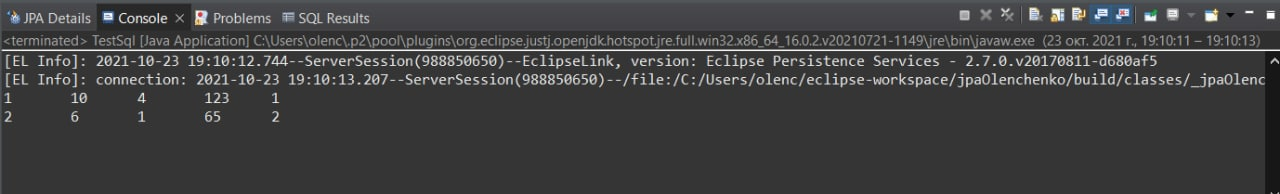


Рисунок 7 – Результат TestSql

**4.2.3** **Завдання для самостійної роботи**

4.2.3.1 Тексти класів моделі

Текст класу Audience зображено в лістингу 9.

Лістинг 9 – Клас Audience

**public** **class** Audience **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L

@Id

@GeneratedValue(strategy=GenerationType.IDENTITY)

**private** **int** id

**private** String area;

**private** String floor;

**private** String number;

//bi-directional many-to-one association to Corps

@ManyToOne(cascade={CascadeType.REMOVE})

@JoinColumn(name="IDCORPS")

**private** Corps corps;

**public** Audience() {

}

**public** **int** getId() {

**return** **this**.id;}

**public** **void** setId(**int** id) {

**this**.id = id;}

**public** String getArea() {

**return** **this**.area;}

**public** **void** setArea(String area) {

**this**.area = area;}

**public** String getFloor() {

**return** **this**.floor;}

**public** **void** setFloor(String floor) {

**this**.floor = floor;}

**public** String getNumber() {

**return** **this**.number;}

**public** **void** setNumber(String number) {

**this**.number = number;}

**public** Corps getCorps() {

**return** **this**.corps;}

**public** **void** setCorps(Corps corps) {

**this**.corps = corps;}}

Текст класу Corps зображено в лістингу 10.

Лістинг 10 – Клас Corps

**public** **class** Corps **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@Id

@GeneratedValue(strategy=GenerationType.IDENTITY)

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

**private** String audience;

**private** String corps;

**private** String furniture;

//bi-directional many-to-one association to Audience

@OneToMany(mappedBy="corps")

**private** List<Audience> audiences;

**public** Corps() {}

**public** **int** getId() {

**return** **this**.id;}

**public** **void** setId(**int** id) {

**this**.id = id;}

**public** String getAudience() {

**return** **this**.audience;}

**public** **void** setAudience(String audience) {

**this**.audience = audience;}

**public** String getCorps() {

**return** **this**.corps;}

**public** **void** setCorps(String corps) {

**this**.corps = corps;}

**public** String getFurniture() {

**return** **this**.furniture;}

**public** **void** setFurniture(String furniture) {

**this**.furniture = furniture;}

**public** List<Audience> getAudiences() {

**return** **this**.audiences;}

**public** **void** setAudiences(List<Audience> audiences) {

**this**.audiences = audiences;}

**public** Audience addAudience(Audience audience) {

getAudiences().add(audience);

audience.setCorps(**this**);

**return** audience;}

**public** Audience removeAudience(Audience audience) {

getAudiences().remove(audience);

audience.setCorps(**null**);

**return** audience;}}

Текст класу Furniture зображено в лістингу 11.

Лістинг 11 – Клас Furniture

**public** **class** Furniture **implements** Serializable {@Id

@GeneratedValue(strategy = IDENTITY)

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

**private** **int** price;

**private** **int** amount\_of\_furniture;

**private** **int** weight;

@ManyToOne(targetEntity = jpaOlenchenko.Audience.**class**, cascade = REMOVE)

**private** Audience audience;

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** Furniture() {

**super**();}

**public** **int** getId() {

**return** **this**.id;}

**public** **void** setId(**int** id) {

**this**.id = id;}

**public** **int** getPrice() {

**return** **this**.price;}

**public** **void** setPrice(**int** price) {

**this**.price = price;}

**public** **int** getAmount\_of\_furniture() {

**return** **this**.amount\_of\_furniture;}

**public** **void** setAmount\_of\_furniture(**int** amount\_of\_furniture) {

**this**.amount\_of\_furniture = amount\_of\_furniture;}

**public** **int** getWeight() {

**return** **this**.weight;}

**public** **void** setWeight(**int** weight) {

**this**.weight = weight;}

**public** Audience getAudience() {

**return** **this**.audience;}

**public** **void** setAudience(Audience audience) {

**this**.audience = audience;}}

4.2.3.2 Реалізація контролера

Текст класу Controler зображено в лістингу 12.

Лістинг 12 – Клас Controller

**public** **class** Controller {

**private** **static** EntityManagerFactory *emf* =

Persistence.createEntityManagerFactory("jpaOlenchenko");

**public** **static** **boolean** tableExist(String tableName) {

// If table exist, it is possible to count number of records.

// If table does not exist, exception occurs.

String testQuery = String.*format*(

"select count(\*) from %s ", tableName);

EntityManager em = *emf*.createEntityManager();

Query q = em.createNativeQuery(testQuery);

**try** {

q.getSingleResult();

**return** **true**;

} **catch** (Exception e) {

**return** **false**;

}**finally** {

em.close();

} }

**public** **static** List<Map<String,Object>> executeQuery(String jpql) { //Output for debuging

System.***out***.println(jpql);

EntityManager em = *emf*.createEntityManager();

Query query = em.createQuery(jpql);

query.setHint(QueryHints.RESULT\_TYPE, ResultType.Map);

@SuppressWarnings("unchecked")

List<Map<String,Object>> list = query.getResultList();

em.close();

**return** list;

}

**public** **static** **void** createTable(String queryClass) {

//This method was not realize, because tables already exist

}

**public** **static** **void** add(String tableName,Map<String,Object> map) {

Object obj = **null**;

**try** {

String queryClass = "query.Query" + tableName;

Class<?> clz = Class.*forName*(queryClass );

Method mth = clz.getDeclaredMethod("createObject",Map.**class**);

obj = mth.invoke(**null**, map);

} **catch** (Exception e1) {

e1.printStackTrace();

JOptionPane.*showMessageDialog*(**null**,"Object create problem");

**return**; }

EntityManager em = *emf*.createEntityManager();

System.***out***.println(obj);

**try** {

em.getTransaction().begin();

em.persist(obj);

em.getTransaction().commit();

} **catch** (Exception e) {

em.getTransaction().rollback();

e.printStackTrace();

} **finally** {

em.close(); }}

**public** **static** **void** edit(String tableName,Map<String,Object> map) {

EntityManager em = *emf*.createEntityManager();

Object obj = **null**;

**try** {

Class<?> clzTable = Class.*forName*("jpaOlenchenko." + tableName);

obj = em.find(clzTable, map.get("id"));

Class<?> clzQuery = Class.*forName*("query.Query" + tableName);

Method mtd = clzQuery.getMethod(

"editObject",Object.**class**, Map.**class**);

obj = mtd.invoke(**null**, obj, map);

} **catch** (Exception e1) {

e1.printStackTrace();

**return**;

}

**if**(obj != **null**)

**try** {

em.getTransaction().begin();

em.merge(obj);

em.getTransaction().commit();

} **catch** (Exception e) {

em.getTransaction().rollback();

e.printStackTrace();

} **finally** {

em.close(); }}

**public** **static** **void** delete(String tableName, Map<String, Object> map) {

**int** id = (**int**)map.get("id");

Class<?> clazz = **null**;

**try** {

clazz = Class.*forName*("jpaOlenchenko." + tableName);

} **catch** (Exception e1) {

e1.printStackTrace();

**return**;}

**if**(clazz != **null**) {

EntityManager em = *emf*.createEntityManager();

**try** {

em.getTransaction().begin();

Object delObj = em.find(clazz, id);

em.remove(delObj);

em.getTransaction().commit();

} **catch** (Exception e) {

e.printStackTrace();

em.getTransaction().rollback();

} **finally** {

em.close(); } } }

**public** **static** Object getObjectById(String tableName, **int** id) {

EntityManager em = *emf*.createEntityManager();

Object obj = **null**;

**try** {

Class<?> clz = Class.*forName*("jpaOlenchenko." + tableName);

obj = em.find(clz, id);

} **catch** (ClassNotFoundException e) {

e.printStackTrace(); }

**return** obj;}

**public** **static** **int** executeUpdate(String query) {

EntityManager em = *emf*.createEntityManager();

Query q = em.createQuery(query);

**try** { em.getTransaction().begin();

**int** n = q.executeUpdate();

em.getTransaction().commit();

**return** n;

} **catch** (Exception e) {

e.printStackTrace();

em.getTransaction().rollback();

} **finally** {

em.close(); }

**return** 0; }}

4.2.3.3 Класи пакету Query

Клас QueryAudience зображено в лістингу 13.

Лістинг 13 – Клас QueryAudience

**public** **class** QueryAudience {

**public** **static** String queryCreate() {

// This method was not realize, because tables already exist

**return** **null**;

}

**public** **static** String queryGetAll() {

**return** "select g.id, g.number, g.area, g.floor," + " g.corps.corps AS corps," + " g.corps.id AS idCorps"

+ " FROM jpaOlenchenko.Audience g " + " ORDER BY corps, g.number";

}

**public** **static** Audience createObject(Map<String, Object> map) {

Audience grd = **new** Audience();

grd.setArea((String) map.get("area"));

grd.setFloor((String) map.get("floor"));

grd.setNumber((String) map.get("number"));

**int** idCorps = (**int**) map.get("idCorps");

Corps corps = (Corps) Controller.*getObjectById*("Corps", idCorps);

grd.setCorps(corps);

**return** grd;}

**public** **static** Object editObject(Object obj, Map<String, Object> map) {

Audience grd = (Audience) obj;

grd.setArea((String) map.get("area"));

grd.setFloor((String) map.get("floor"));

grd.setNumber((String) map.get("number"));

**int** idCorps = (**int**) map.get("idCorps");

Corps corps = (Corps) Controller.*getObjectById*("Corps", idCorps);

grd.setCorps(corps);

**return** grd; }}

Клас QueryCorps зображено в лістингу 14.

Лістинг 14 – Клас QueryCorps

**public** **class** QueryCorps {

**public** **static** String queryCreate() {

// This method was not realize, because tables already exist

**return** **null**;}

**public** **static** String queryGetAll() {

**return** "select c.id, c.audience, c.corps, c.furniture from Corps c"; }

**public** **static** Corps createObject(Map<String, Object> map) {

Corps grd = **new** Corps();

grd.setAudience((String) map.get("audience"));

grd.setCorps((String) map.get("corps"));

grd.setFurniture((String) map.get("furniture"));

**return** grd;}

**public** **static** Object editObject(Object obj, Map<String, Object> map) {

Corps grd = (Corps) obj;

grd.setAudience((String) map.get("area"));

grd.setCorps((String) map.get("floor"));

grd.setFurniture((String) map.get("number"));

**return** grd;}}

Клас QueryFurniture зображено в лістингу 15.

Лістинг 15 – Клас QueryFurniture

**public** **class** QueryFurniture {

**public** **static** String queryCreate() {

// This method was not realize, because tables already exist

**return** **null**;}

**public** **static** String queryGetAll() {

**return** "select g.id, g.price, g.amount\_of\_furniture, g.weight, g.audience.number AS audience," +

" g.audience.id AS idAudience FROM jpaOlenchenko.Furniture g ORDER BY audience, g.price";

}

**public** **static** Furniture createObject(Map<String, Object> map) {

Furniture grd = **new** Furniture();

grd.setPrice((**int**) map.get("price"));

grd.setAmount\_of\_furniture((**int**) map.get("amount\_of\_furniture"));

grd.setWeight((**int**) map.get("weight"));

**int** idAudience= (**int**) map.get("idAudience");

Audience audience = (Audience) Controller.*getObjectById*("Audience", idAudience);

grd.setAudience(audience);

**return** grd;

}

**public** **static** Object editObject(Object obj, Map<String, Object> map) {

Furniture grd = (Furniture) obj;

grd.setPrice((**int**) map.get("price"));

grd.setAmount\_of\_furniture((**int**) map.get("amount\_of\_furniture"));

grd.setWeight((**int**) map.get("weight"));

**int** idAudience= (**int**) map.get("idAudience");

Audience audience = (Audience) Controller.*getObjectById*("Audience", idAudience);

grd.setAudience(audience);

**return** grd;}

**public** **static** String queryDelById(**int** id) {

**return** "DELETE FROM Furniture WHERE id = " + id;}}

4.2.3.4 Створення класу DlgFurniture

Діалог Furniture зображено на рисунку 8.

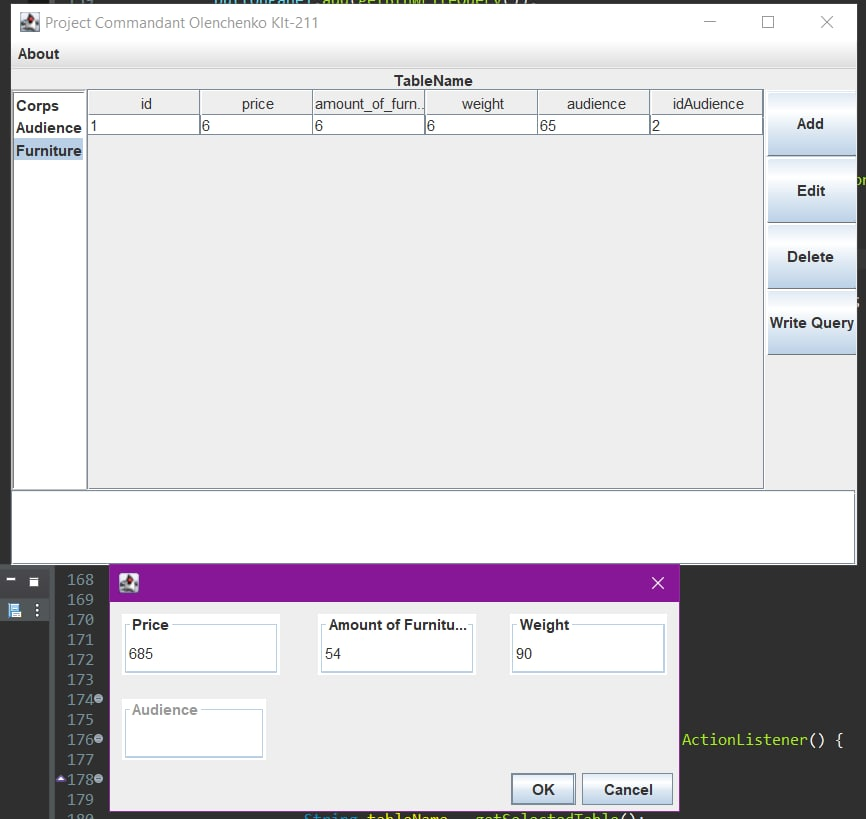


Рисунок 8 – Діалог Furniture

**4.3 Висновки**

В результаті виконання лабораторної роботи ми отримали такі знання і навички:

– ознайомитися з технологією об’єктно-реляційного відображення;

– навчитися працювати з базою даних на рівні об’єктів та створювати запити до бази, використовуючи мову запитів JPQL;

– отримати навички створення програми для роботи з реляційною базою даних через об’єктно-реляційне відображення.