

**Міністерство освіти і науки України
Національний аерокосмічний університет
ім. М.Є. Жуковського**

Кафедра 503

Лабораторна робота № 5
З дисципліни «Кросплатформенні технології»
Тема: «Ознайомлення з JDBC (Java DataBase Connectivity) API.»

Виконав:
студент групи 535 ст 1 Гужва М.А.

Харків 2020

Мета роботи:

1. Ознайомлення з JDBC (Java DataBase Connectivity) API. Засвоєння класів Connection, DatabaseMetaData, Statement, ResultSet пакета java.sql.
2. Вивчення і застосування шаблонів проектування як типових рішень організації і взаємодії об'єктів певної структури і поведінки.

Хід лабораторної роботи

Використання баз даних в кросплатформених програмних додатках є одним з ключових тенденцій розробки останніх десятиріч. Розподілене керування масивами даних, швидкий пошук та контроль даних, обумовлюють використання технологій баз даних в сучасних проектах. Java – одна з потужних кросплатформених мов програмування, що підтримує роботу з базами даних на основі JDBC.

Java DataBase Connectivity (англ. Java DataBase Connectivity — з'єднання з базами даних на Java), скорочено JDBC) — прикладний програмний інтерфейс Java, який визначає методи, з допомогою яких програмне забезпечення на Java здійснює доступ до бази даних. JDBC — це платформо-незалежний промисловий стандарт взаємодії Java-застосунків з різноманітними СУБД, реалізований у вигляді пакета java.sql, що входить до складу Java SE.

В основі JDBC лежить концепція так званих драйверів, що дозволяють отримувати з'єднання з базою даних по спеціально описаному URL. Драйвери можуть завантажуватись динамічно (під час роботи програми). Завантажившись, драйвер сам реєструє себе й викликається автоматично, коли програма вимагає URL, що містить протокол, за який драйвер «відповідає».

Перевагами JDBC вважаються:

- 1) Легкість розробки: розробник може не знати специфіки бази даних, з якою працює;
- 2) Код не змінюється, якщо розробнику необхідний перехід на іншу базу даних;
- 3) Не треба встановлювати громіздку клієнтську програму;
- 4) До будь-якої бази можна під'єднатись через легко описуваний URL.

JDBC API містить два основні типи інтерфейсів: перший — для розробників застосунків і другий (нижчого рівня) — для розробників драйверів.

З'єднання з базою даних описується класом, що реалізує інтерфейс `java.sql.Connection`.

Маючи з'єднання з базою даних, можна створювати об'єкти типу `Statement`, використовувані для здійснення запитів до бази даних на мові SQL.

Існують такі види типів `Statement`, що відрізняються своїм призначенням:

`java.sql.Statement` — `Statement` загального призначення;

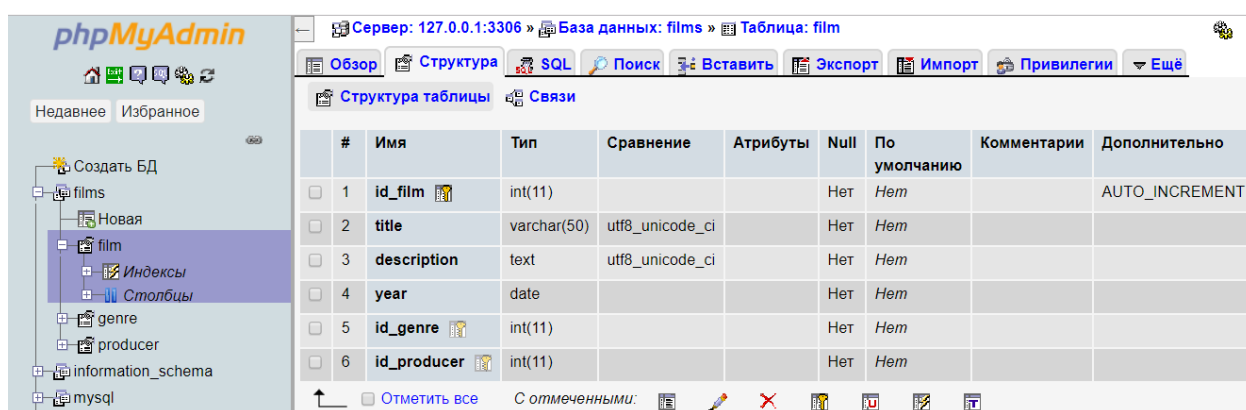
`java.sql.PreparedStatement` — `Statement`, що служить для здійснення запитів, котрі містять підставні параметри (позначаються символом '?' у тілі запиту);

`java.sql.CallableStatement` — `Statement`, призначений для виклику збережених процедур.

Клас `java.sql.ResultSet` дозволяє легко обробляти результати запитів.

Для створення додатку роботи з БД необхідно спочатку розробити схему БД. Для зручності реалізації БД оберемо реляційну БД MySQL та `phpMyAdmin` в якості СКБД, що може входити в популярний пакет `OpenServer` для розробників веб-додатків.

На рис. 1 наведено створення таблиць бази даних `films`, предметною областю якою стануть кінокартини. До мінімального набору таблиць входить – `film` – таблиця фільмів, `director` – дані режисерів, `genre` – дані жанрів.



#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Комментарии	Дополнительно
<input type="checkbox"/>	1	id_film	int(11)		Нет	Нет		AUTO_INCREMENT
<input type="checkbox"/>	2	title	varchar(50)	utf8_unicode_ci	Нет	Нет		
<input type="checkbox"/>	3	description	text	utf8_unicode_ci	Нет	Нет		
<input type="checkbox"/>	4	year	date		Нет	Нет		
<input type="checkbox"/>	5	id_genre	int(11)		Нет	Нет		
<input type="checkbox"/>	6	id_producer	int(11)		Нет	Нет		

Рисунок 1 – Структура таблиці `film`

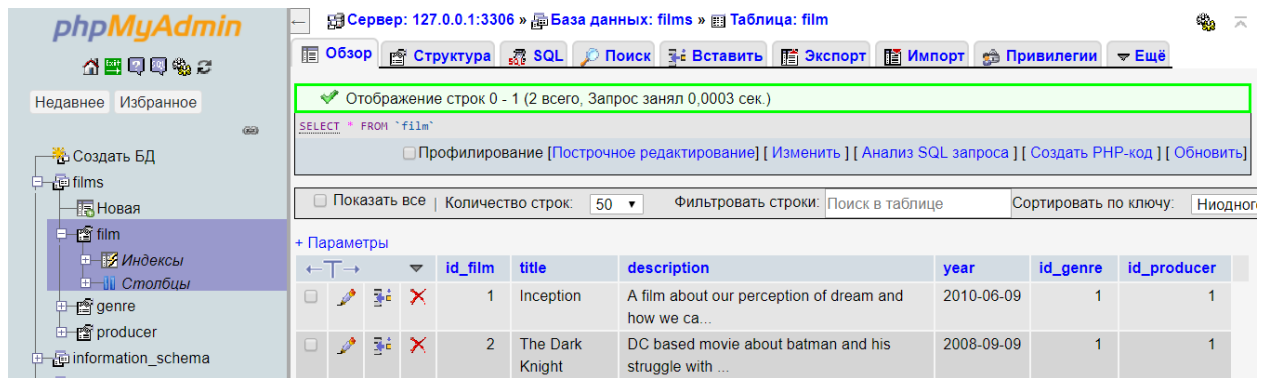


Рисунок 2 – Заповнення таблиць даними

На рис. 3 зображено фрагмент коду програми, що відповідає за опис БД, а саме хост – localhost:3306, username, password - `root`, як було встановлено в phpMyAdmin.



Рисунок 3 – Фрагмент коду опису бази даних

На рис. 4 наведено порядок роботи з БД за допомогою JDBC, а саме – налаштування класу DriverManager, створення об'єкту Connection, створення та виконання запиту – executeQuery().



Рисунок 4 – Порядок роботи з БД MySQL за допомогою JDBC

Результат розробки ПЗ – невеликий консольний додаток (рис.5), що підключається до БД на phpMyAdmin, та дозволяє зчитувати дані з таблиці film та записувати дані за допомогою методу executeUpdate(insert_statement).

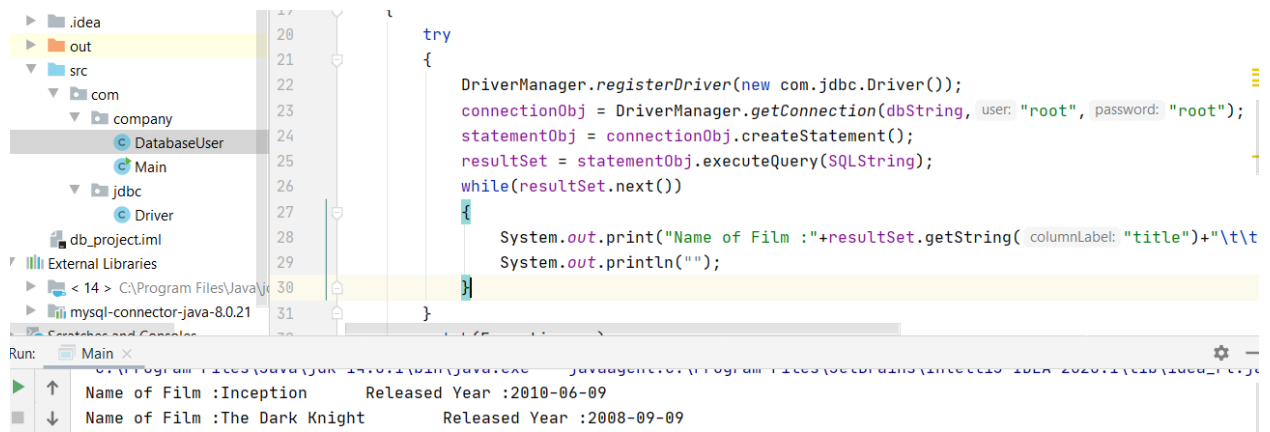


Рисунок 5 – Приклад роботи додатку

Завдання 2 – Огляд патернів проектування

Абстрактна фабрика – це породжуючий патерн.

Призначення: створення об'єктів, які належать до однієї групи, шляхом абстрагування.

Для того, щоби краще зрозуміти що це, розглянемо наочний приклад.

Припустимо, є рок-група, в якій є кілька учасників. Один з них грає на акустичній гітарі, другий – на електрогітарі, а третій відповідно від бажання на електричних барабанах або на акустичних.

Необхідно дати кожному музиканту той музичний інструмент, який йому потрібно. Це означає, що магазин – абстрактна фабрика, бо може надати потрібний інструмент, не зважаючи на його тип – електричний чи акустичний.

Коли гітарист просить акустичну гітару, магазин видає йому її. Отже, електричний інструмент і акустичний – це вже не абстракція, а конкретні фабрики, які містять конкретні інструменти – електричні та акустичні.

Алгоритм створення реалізації такого патерна можна описати наступним чином:

1. Створити класи, об'єкти яких є сімейством, що необхідно розділити на фабрики;
2. Створити абстрактний клас (абстрактну фабрику) з чистими віртуальними функціями, які повертатимуть конкретний об'єкт з конкретної фабрики;
3. Створити конкретні фабрики – класи, що успадковані від абстрактного та в яких реалізовані чисті віртуальні функції, що повертають конкретний об'єкт;
4. Отримати потрібний об'єкт.

Приклад використання шаблону Abstract Factory

```
abstract class
AbstractProductA{

    public abstract void operationA1();
    public abstract void operationA2();
}

class ProductA1 extends AbstractProductA{
    ProductA1(String arg){
        System.out.println("Hello "+arg);
    } // Implement the code here
    public void operationA1() { };
    public void operationA2() { };
}

class ProductA2 extends AbstractProductA{
    ProductA2(String arg){
        System.out.println("Hello "+arg);
    } // Implement the code here
    public void operationA1() { };
    public void operationA2() { };
}

abstract class AbstractProductB{
    //public abstract void operationB1();
    //public abstract void operationB2();
}

class ProductB1 extends AbstractProductB{
    ProductB1(String arg){
        System.out.println("Hello "+arg);
    } // Implement the code here
}

class ProductB2 extends AbstractProductB{
    ProductB2(String arg){
        System.out.println("Hello "+arg);
    } // Implement the code here
}

abstract class AbstractFactory{
    abstract AbstractProductA createProductA();
    abstract AbstractProductB createProductB();
}

class ConcreteFactory1 extends AbstractFactory{
    AbstractProductA createProductA(){
        return new ProductA1("ProductA1");
    }
    AbstractProductB createProductB(){
        return new ProductB1("ProductB1");
    }
}

class ConcreteFactory2 extends AbstractFactory{
    AbstractProductA createProductA(){
```

```

        return new ProductA2("ProductA2");
    }
    AbstractProductB createProductB(){
        return new ProductB2("ProductB2");
    }
}
//Factory creator - an indirect way of instantiating the factories
class FactoryMaker{
    private static AbstractFactory pf=null;
    static AbstractFactory getFactory(String choice){
        if(choice.equals("a")){
            pf=new ConcreteFactory1();
        }else if(choice.equals("b")){
            pf=new ConcreteFactory2();
        } return pf;
    }
}
// Client
public class Client{
    public static void main(String args[]){
        AbstractFactory pf=FactoryMaker.getFactory("a");
        AbstractProductA product=pf.createProductA();
        //more function calls on product
    }
}

```

Висновки: в ході виконання лабораторної роботи було засвоєно порядок роботи з базами даних за допомогою кросплатформених технологій Java та драйверів JDBC на прикладі підключення та виконання запитів SELECT та INSERT/DELETE до таблиць MySQL phpMyAdmin БД films з додатку Java. Окремо було засвоєно особливості застосування патернів проектування, зокрема – Abstract Factory. Даний патерн зручно використовувати при реалізації моделей з багатьма типами об'єктів, що мають спільні характеристики.