

Отчет по лабораторной работе 4

Студент: Михайлов Евгений

Группа: ПИМ-22

1. Постановка задачи

В процессе выполнения лабораторной работы необходимо выполнить следующие задачи:

1. ClassLoader

- Ознакомиться с руководством по загрузке классов и ClassLoader
- Продемонстрировать работу своего загрузчика классов
- Определить разницу между своей и стандартной реализацией

2. JDBC

- Установить соединение с БД с помощью JDBC драйвера
- Создать таблицу БД с помощью JDBC
- Реализовать CRUD-методы для одной таблицы
- Реализовать несколько запросов в БД, которые должны выполняться в рамках одной транзакции

3. Exception

- Обернуть методы для работы с БД в try/catch с откатом транзакций и закрытием соединения
- Продемонстрировать в программе откат транзакции

2. Разработка задачи

2.1 Структура проекта

Проект разделен на следующие директории:

docs

Данная документация

lib

Подключенные библиотеки

src

Исходный код лабораторной работы

src/classloader Задание 1

src/db

Задание 2 и 3

3. Информация о реализации

3.1 Задание 1

Для выполнения первого задания созданы следующие классы:

- CustomClassLoader - реализация своего класслоадера.
- DemoClass - класс, который будет загружаться своим класслоадером.

В класс Main был добавлен метод testClassLoader, демонстрирующий работу класслоадера.

Листинг 1. CustomClassLoader

```
package classloader;

import java.io.ByteArrayOutputStream;
import java.io.InputStream;

public class CustomClassLoader extends ClassLoader {

    private byte[] loadClassBytes(String name) {
        InputStream in = getClass().getClassLoader().getResourceAsStream(name.replace(".", "/") + ".class");
        ByteArrayOutputStream stream = new ByteArrayOutputStream();
        int length = 0;

        try {
            assert in != null;
            length = in.read();
            while (length != -1) {
                stream.write(length);
                length = in.read();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return stream.toByteArray();
    }

    @Override
    public Class<?> findClass(String name) {
        byte[] b = loadClassBytes(name);
        return defineClass(name, b, 0, b.length);
    }
}
```

Листинг 2. DemoClass

```
package classloader;

public class DemoClass {
    public void test1() {
        System.out.println("DemoClass loaded successfully");
    }
    public void helloWorld(String name) {
        System.out.println("Hello "+name+" from DemoClass");
    }
}
```

Листинг 3. Main

```
private static void testClassLoader() throws InstantiationException,
IllegalAccessException, NoSuchMethodException, InvocationTargetException {
    CustomClassLoader customClassLoader = new CustomClassLoader();
    Class<?> cl = customClassLoader.findClass("classloader.DemoClass");
    Object ob = cl.newInstance();
    Method method = cl.getMethod("test1");
    method.invoke(ob);
    method = cl.getMethod("helloWorld", String.class);
    method.invoke(ob, "ClassLoader");
}
```

Отличием собственной реализации класслоадера и стандартной реализацией является переопределенный метод `findClass`, а также тем что, что классы, загруженные через стандартные загрузчики Java не будут видеть этот класс.

3.2 Задание 2 и 3

Для выполнения этого задания добавим в проект класс DB и библиотеку JDBC(sqlite), так же методы `testDB` и `dbPrintData` в класс Main.

Реализовано:

- Установка соединения с базой данных с использованием JDBC драйвера.
- Создание таблицы
- Вставка строк данных в таблицу.
- Обновление данных в таблице.
- Удаление строк из таблицы.
- Отключение от базы данных

Функции находятся внутри try блока, в случае ошибки соединение и statement закрываются.

Листинг 4. DB

```
package db;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DB {
    private Connection connection = null;
    private Statement statement = null;

    public void connect() {
```

```

    try {
        connection = DriverManager.getConnection("jdbc:sqlite::memory:");
        connection.setAutoCommit(false);
        statement = connection.createStatement();
        statement.setQueryTimeout(30); // set timeout to 30 sec.
    } catch (SQLException e) {
        this.close();
        e.printStackTrace();
    }
}

public void createTable(String name, String params) {
    try {
        statement.executeUpdate("CREATE TABLE IF NOT EXISTS " + name + " " +
params);
    } catch (SQLException e) {
        this.rollback();
        this.close();
        e.printStackTrace();
    }
}

public void update(String query) {
    try {
        statement.executeUpdate(query);
    } catch (SQLException e) {
        this.rollback();
        this.close();
        e.printStackTrace();
    }
}

public ResultSet select(String query) {
    try {
        return statement.executeQuery(query);
    } catch (SQLException e) {
        this.close();
        return null;
    }
}

public void commit() {
    try {
        connection.commit();
    } catch (SQLException e) {
        this.rollback();
        this.close();
        e.printStackTrace();
    }
}

```

```
public void rollback() {  
    try {  
        connection.rollback();  
    } catch (SQLException e) {  
        this.close();  
        e.printStackTrace();  
    }  
}  
  
public void close() {  
    try {  
        if (connection != null) {  
            connection.close();  
            connection = null;  
        }  
        if (statement != null) {  
            statement.close();  
            statement = null;  
        }  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}  
}
```

```

private static void dbPrintData(DB db, ResultSet rs) {
    if (rs != null) {
        try {
            System.out.println("-----");
            while (rs.next()) {
                System.out.println("ID="+rs.getInt(1)+" NAME="+rs.getString(2) + "
COUNTER=" + rs.getInt(3));
            }
        } catch (SQLException e) {
            db.close();
            e.printStackTrace();
        }
    }
}

private static void testDB() {
    DB db = new DB();
    db.connect();

    db.createTable("test", "(id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT
NULL, counter INTEGER DEFAULT 0)");
    db.update("INSERT INTO test (name, counter) VALUES ('test', 100)");
    db.update("INSERT INTO test (name, counter) VALUES ('test1', 150)");
    db.commit();

    ResultSet rs = db.select("SELECT * from test");
    dbPrintData(db, rs);

    db.update("UPDATE test SET counter = 300 WHERE name = 'test1'");
    rs = db.select("SELECT * from test");
    dbPrintData(db, rs);

    db.update("DELETE FROM test WHERE name = 'test1'");
    rs = db.select("SELECT * from test");
    dbPrintData(db, rs);
    db.commit();

    //Откат транзакции
    db.update("DELETE FROM test WHERE name = 'test'");
    db.rollback();
    rs = db.select("SELECT * from test");
    dbPrintData(db, rs);

    //Некорректный запрос
    db.update("DELETE FROM test WHERE name = 'test1'");

    db.close();
}

```

3. Результаты выполнения

В результате выполнения лабораторной работы получены следующие java классы:

- Main - главный класс приложения
- DemoClass, CustomClassLoader - для демонстрации собственного загрузчика классов.
- DB для демонстрации подключения к базе данных через драйвер JDBC, создания таблицы, выполнения CRUD-методов, в том числе завернутых в try/catch, а также для демонстрации отката транзакции.

Результат запуска Main

```
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" "-javaagent:C:\Program Files\Android\Android Studio\lib\idea_...
.0.0.jar Main
DemoClass loaded successfully
Hello Classloader from DemoClass
-----
ID=1 NAME=test COUNTER=100
ID=2 NAME=test1 COUNTER=150
-----
ID=1 NAME=test COUNTER=100
ID=2 NAME=test1 COUNTER=300
-----
ID=1 NAME=test COUNTER=100
-----
ID=1 NAME=test COUNTER=100
org.sqlite.SQLiteException: Create breakpoint : [SQLITE_ERROR] SQL error or missing database (unrecognized token: "'test1'")
    at org.sqlite.core.DB.newSQLException(DB.java:1135)
    at org.sqlite.core.DB.newSQLException(DB.java:1146)
    at org.sqlite.core.DB.throwex(DB.java:1106)
    at org.sqlite.core.NativeDB._exec_utf8(Native Method)
    at org.sqlite.core.NativeDB._exec(NativeDB.java:89)
    at org.sqlite.jdbc3.JDBC3Statement.lambda$executeLargeUpdate$2(JDBC3Statement.java:128)
    at org.sqlite.jdbc3.JDBC3Statement.withConnectionTimeout(JDBC3Statement.java:454)
    at org.sqlite.jdbc3.JDBC3Statement.executeLargeUpdate(JDBC3Statement.java:115)
    at org.sqlite.jdbc3.JDBC3Statement.executeUpdate(JDBC3Statement.java:102)
    at db.DB.update(DB.java:37)
    at Main.testDB(Main.java:63)
    at Main.main(Main.java:70)

Process finished with exit code 0
```

4. Вывод

В результате выполнения лабораторной работы получены навыки по созданию собственной реализации загрузчика классов. А также по работе с драйвером JDBC, а именно - подключении к базе данных, создании таблицы, выполнении CRUD-методов, использовании транзакций.