МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА «ПРОГРАМНОЇ ІНЖЕНЕРІЇ ТА ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ ІМЕНІ А. В. ДАБАГЯНА»

ЗВІТ

З лабораторної роботи №5

«РЕФЛЕКСІЯ. СТВОРЕННЯ ЗАСТОСУНКІВ БАЗ ДАНИХ»

ВИКОНАВ:

студент групи КН-422ч

Максим БЕЛОШИЦЬКИЙ

ПЕРЕВІРИВ:

Асистент каф. ПІІТУ

Олексій КОНДРАТОВ

Харків – 2023

**ЗМІСТ**

[МЕТА 3](#_Toc167537476)

[Завдання 1.1 4](#_Toc167537477)

[Код завдання 1.1: 4](#_Toc167537478)

[db\_creation.sql 4](#_Toc167537479)

[ConferenceManager.java 5](#_Toc167537480)

[Conference.java 13](#_Toc167537481)

[Meeting.java 16](#_Toc167537482)

[Тестування за допомогою JUnit: 19](#_Toc167537483)

[ConferenceTest.java 19](#_Toc167537484)

[MeetingTest.java 21](#_Toc167537485)

[Результати виконання програми 23](#_Toc167537486)

[export.json 27](#_Toc167537487)

[Завдання 1.2 29](#_Toc167537488)

[Код завдання 1.2: 29](#_Toc167537489)

[FieldViewer.java 29](#_Toc167537490)

[Тестування за допомогою JUnit: 30](#_Toc167537491)

[FieldViewerTest.java 30](#_Toc167537492)

[Результати виконання програми 31](#_Toc167537493)

[Завдання 1.3 32](#_Toc167537494)

[Код завдання 1.3: 33](#_Toc167537495)

[ConferenceManagerGUI.java 33](#_Toc167537496)

[Тестування за допомогою JUnit: 39](#_Toc167537497)

[ConferenceManagerGUITest.java 39](#_Toc167537498)

[Результати виконання програм 40](#_Toc167537499)

[Завдання 1.4 43](#_Toc167537500)

[Код завдання 1.4: 43](#_Toc167537501)

[AnnotationInfo.java 43](#_Toc167537502)

[Тестування за допомогою JUnit: 44](#_Toc167537503)

[AnnotationInfoTest.java 44](#_Toc167537504)

[Результати виконання програми 45](#_Toc167537505)

[Завдання для самоконтролю 47](#_Toc167537506)

[Код завдання 2.1: 47](#_Toc167537507)

[Результати виконання програми 48](#_Toc167537508)

[Код завдання 2.2: 48](#_Toc167537509)

[Код завдання 2.3: 49](#_Toc167537510)

[ВИСНОВОК 50](#_Toc167537511)

# **МЕТА**

Мета цієї лабораторної роботи полягає в розробці консольного застосунку баз даних для роботи з сутностями, що відповідають індивідуальним завданням попередніх лабораторних робіт. Застосунок повинен працювати з реляційною базою даних, використовуючи СУБД MySQL або іншу, згідно з узгодженням з викладачем.

Після створення таблиць у базі даних необхідно виконати наступні дії:

1. Імпортувати дані в базу даних з JSON-файлу.
2. Відобразити дані обох таблиць.
3. Здійснити пошук записів за ознаками, визначеними в третій лабораторній роботі.
4. Виконати сортування записів за ознаками, визначеними в четвертій лабораторній роботі.
5. Додати новий запис в таблицю.
6. Видалити певні записи з таблиці.
7. Експортувати дані з бази даних в JSON-файл.

Додатково, створити консольний застосунок, в якому користувач може вводити ім'я класу і отримувати інформацію про всі поля цього класу, включаючи закриті й захищені.

# Завдання 1.1

Створити консольний застосунок баз даних для роботи з сутностями індивідуальних завдань попередніх лабораторних робіт.

Дані про сутності предметної області повинні зберігатися у реляційній базі даних. Слід застосувати СУБД MySQL (або іншу СУБД, за погодженням з викладачем). У консольному застосунку слід створити дві таблиці реляційної бази даних, відповідно для першої та другої сутності індивідуального завдання (перша та друга таблиця).

Після створення таблиць необхідно виконати такі дії:

* імпорт в базу даних з JSON-файлу;
* відображення даних обох таблиць;
* пошук за ознаками, визначеними в [третій лабораторній роботі](http://iwanoff.inf.ua/java_ua/LabTraining03.html#Tasks) курсу "Основи програмування Java";
* здійснення сортування за ознаками, визначеними в [четвертій лабораторній роботі](http://iwanoff.inf.ua/java_ua/LabTraining04.html#Tasks) курсу "Основи програмування Java";
* додавання нового запису в таблицю;
* видалення певних записів з таблиці;
* експорт з бази даних в JSON-файл.

## Код завдання 1.1:

### db\_creation.sql

CREATE DATABASE IF NOT EXISTS conference\_manager;

USE conference\_manager;

CREATE TABLE IF NOT EXISTS conferences (

    id *INT* AUTO\_INCREMENT PRIMARY KEY,

    name *VARCHAR*(255) NOT NULL,

    location *VARCHAR*(255) NOT NULL

);

CREATE TABLE IF NOT EXISTS meetings (

    id *INT* AUTO\_INCREMENT PRIMARY KEY,

    conference\_id *INT* NOT NULL,

    topic *VARCHAR*(255) NOT NULL,

*date* *VARCHAR*(255) NOT NULL,

    participants *INT* NOT NULL,

    FOREIGN KEY (conference\_id) REFERENCES conferences(id)

);

### ConferenceManager.java

package *edu.lab04\_ind*;

import *java.sql.\**;

import *java.util.List*;

import *java.util.Scanner*;

import *java.io.\**;

import *com.google.gson.Gson*;

import *com.google.gson.GsonBuilder*;

import *java.util.ArrayList*;

*/\*\**

*\* The ConferenceManager class is responsible for managing conferences and meetings.*

*\* It provides functionality to add, delete, display, search, and sort conferences and meetings.*

*\*/*

*public* *class* ConferenceManager {

*private* *static* *final* *String* URL = "jdbc:mysql://localhost:3306/ConferenceDB";

*private* *static* *final* *String* USER = "root";

*private* *static* *final* *String* PASSWORD = "228228R@x";

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

        try (*Connection* conn = DriverManager.getConnection(URL, USER, PASSWORD)) {

            createTables(conn);

*Scanner* scanner = new Scanner(System.in);

            while (true) {

                System.out.println("\nPlease choose an action:");

                System.out.println("1. Add a new conference");

                System.out.println("2. Add a meeting to a conference");

                System.out.println("3. Delete a conference");

                System.out.println("4. Delete a meeting");

                System.out.println("5. Display all conferences and meetings");

                System.out.println("6. Search for meetings");

                System.out.println("7. Sort meetings");

                System.out.println("8. Import conferences and meetings from JSON");

                System.out.println("9. Export conferences and meetings to JSON");

                System.out.println("10. Exit");

*int* choice = scanner.nextInt();

                scanner.nextLine(); *// Consume newline*

                switch (choice) {

                    case 1:

                        addConference(conn, scanner);

                        break;

                    case 2:

                        addMeetingToConference(conn, scanner);

                        break;

                    case 3:

                        deleteConference(conn, scanner);

                        break;

                    case 4:

                        deleteMeeting(conn, scanner);

                        break;

                    case 5:

                        displayConferencesAndMeetings(conn);

                        break;

                    case 6:

                        searchMeetings(conn, scanner);

                        break;

                    case 7:

                        sortMeetings(conn, scanner);

                        break;

                    case 8:

                        importFromJSON(conn, scanner);

                        break;

                    case 9:

                        exportToJSON(conn, scanner);

                        break;

                    case 10:

                        System.out.println("Exiting...");

                        return;

                    default:

                        System.out.println("Invalid choice. Please try again.");

                }

            }

        } catch (*SQLException* *e*) {

            e.printStackTrace();

        }

    }

*private* *static* *void* createTables(*Connection* *conn*) *throws* *SQLException* {

*String* conferencesTableSQL = "CREATE TABLE IF NOT EXISTS Conferences (" +

                "id INT AUTO\_INCREMENT PRIMARY KEY," +

                "name VARCHAR(255) NOT NULL," +

                "place VARCHAR(255) NOT NULL)";

*String* meetingsTableSQL = "CREATE TABLE IF NOT EXISTS Meetings (" +

                "id INT AUTO\_INCREMENT PRIMARY KEY," +

                "conference\_id INT," +

                "date DATE NOT NULL," +

                "name VARCHAR(255) NOT NULL," +

                "numberOfParticipants INT NOT NULL," +

                "FOREIGN KEY (conference\_id) REFERENCES Conferences(id) ON DELETE CASCADE)";

        try (*Statement* stmt = conn.createStatement()) {

            stmt.executeUpdate(conferencesTableSQL);

            stmt.executeUpdate(meetingsTableSQL);

        }

    }

*private* *static* *void* addConference(*Connection* *conn*, *Scanner* *scanner*) *throws* *SQLException* {

        System.out.println("Enter conference name:");

*String* conferenceName = scanner.nextLine();

        System.out.println("Enter conference place:");

*String* conferencePlace = scanner.nextLine();

*String* insertConferenceSQL = "INSERT INTO Conferences (name, place) VALUES (?, ?)";

        try (*PreparedStatement* stmt = conn.prepareStatement(insertConferenceSQL, Statement.RETURN\_GENERATED\_KEYS)) {

            stmt.setString(1, conferenceName);

            stmt.setString(2, conferencePlace);

            stmt.executeUpdate();

*ResultSet* rs = stmt.getGeneratedKeys();

*int* conferenceId = 0;

            if (rs.next()) {

                conferenceId = rs.getInt(1);

            }

            System.out.println("Conference added successfully with ID: " + conferenceId);

        }

    }

*private* *static* *void* addMeetingToConference(*Connection* *conn*, *Scanner* *scanner*) *throws* *SQLException* {

        displayConferences(conn);

        System.out.println("Enter the ID of the conference to which you want to add a meeting:");

*int* conferenceId = scanner.nextInt();

        scanner.nextLine(); *// Consume newline*

        System.out.println("Enter meeting date (YYYY-MM-DD):");

*String* date = scanner.nextLine();

        System.out.println("Enter meeting name:");

*String* meetingName = scanner.nextLine();

        System.out.println("Enter number of participants:");

*int* numberOfParticipants = scanner.nextInt();

        scanner.nextLine(); *// Consume newline*

*String* insertMeetingSQL = "INSERT INTO Meetings (conference\_id, date, name, numberOfParticipants) VALUES (?, ?, ?, ?)";

        try (*PreparedStatement* stmt = conn.prepareStatement(insertMeetingSQL)) {

            stmt.setInt(1, conferenceId);

            stmt.setDate(2, Date.valueOf(date));

            stmt.setString(3, meetingName);

            stmt.setInt(4, numberOfParticipants);

            stmt.executeUpdate();

            System.out.println("Meeting added successfully to conference with ID: " + conferenceId);

        }

    }

*private* *static* *void* deleteConference(*Connection* *conn*, *Scanner* *scanner*) *throws* *SQLException* {

        displayConferences(conn);

        System.out.println("Enter the ID of the conference to delete:");

*int* conferenceId = scanner.nextInt();

        scanner.nextLine(); *// Consume newline*

*String* deleteMeetingsSQL = "DELETE FROM Meetings WHERE conference\_id = ?";

*String* deleteConferenceSQL = "DELETE FROM Conferences WHERE id = ?";

        try (*PreparedStatement* deleteMeetingsStmt = conn.prepareStatement(deleteMeetingsSQL);

*PreparedStatement* deleteConferenceStmt = conn.prepareStatement(deleteConferenceSQL)) {

            deleteMeetingsStmt.setInt(1, conferenceId);

            deleteMeetingsStmt.executeUpdate();

            deleteConferenceStmt.setInt(1, conferenceId);

*int* deletedRows = deleteConferenceStmt.executeUpdate();

            if (deletedRows > 0) {

                System.out.println("Conference with ID " + conferenceId + " deleted successfully.");

            } else {

                System.out.println("No conference found with ID " + conferenceId);

            }

        }

    }

*private* *static* *void* deleteMeeting(*Connection* *conn*, *Scanner* *scanner*) *throws* *SQLException* {

        System.out.println("Enter the ID of the meeting to delete:");

*int* meetingId = scanner.nextInt();

        scanner.nextLine(); *// Consume newline*

*String* deleteMeetingSQL = "DELETE FROM Meetings WHERE id = ?";

        try (*PreparedStatement* stmt = conn.prepareStatement(deleteMeetingSQL)) {

            stmt.setInt(1, meetingId);

*int* deletedRows = stmt.executeUpdate();

            if (deletedRows > 0) {

                System.out.println("Meeting with ID " + meetingId + " deleted successfully.");

            } else {

                System.out.println("No meeting found with ID " + meetingId);

            }

        }

    }

*private* *static* *void* displayConferences(*Connection* *conn*) *throws* *SQLException* {

*String* query = "SELECT \* FROM Conferences";

        try (*Statement* stmt = conn.createStatement(); *ResultSet* rs = stmt.executeQuery(query)) {

            System.out.println("\nConferences:");

            while (rs.next()) {

*int* conferenceId = rs.getInt("id");

*String* conferenceName = rs.getString("name");

*String* conferencePlace = rs.getString("place");

                System.out.println("ID: " + conferenceId + ", Name: " + conferenceName + ", Place: " + conferencePlace);

            }

        }

    }

*private* *static* *void* displayConferencesAndMeetings(*Connection* *conn*) *throws* *SQLException* {

*String* query = "SELECT \* FROM Conferences";

        try (*Statement* stmt = conn.createStatement(); *ResultSet* rs = stmt.executeQuery(query)) {

            System.out.println("\nConferences and Meetings:");

            while (rs.next()) {

*int* conferenceId = rs.getInt("id");

*String* conferenceName = rs.getString("name");

*String* conferencePlace = rs.getString("place");

                System.out.println("Conference ID: " + conferenceId + ", Name: " + conferenceName + ", Place: " + conferencePlace);

*String* meetingQuery = "SELECT \* FROM Meetings WHERE conference\_id = ?";

                try (*PreparedStatement* meetingStmt = conn.prepareStatement(meetingQuery)) {

                    meetingStmt.setInt(1, conferenceId);

*ResultSet* meetingRs = meetingStmt.executeQuery();

                    while (meetingRs.next()) {

*int* meetingId = meetingRs.getInt("id");

*String* date = meetingRs.getDate("date").toString();

*String* meetingName = meetingRs.getString("name");

*int* numberOfParticipants = meetingRs.getInt("numberOfParticipants");

                        System.out.println("\tMeeting ID: " + meetingId + ", Date: " + date + ", Name: " + meetingName + ", Participants: " + numberOfParticipants);

                    }

                }

            }

        }

    }

*private* *static* *void* searchMeetings(*Connection* *conn*, *Scanner* *scanner*) *throws* *SQLException* {

        System.out.println("Enter search criteria:");

*String* criteria = scanner.nextLine();

*String* query = "SELECT \* FROM Meetings WHERE name LIKE ?";

        try (*PreparedStatement* stmt = conn.prepareStatement(query)) {

            stmt.setString(1, "%" + criteria + "%");

*ResultSet* rs = stmt.executeQuery();

            System.out.println("\nSearch Results:");

            while (rs.next()) {

*int* meetingId = rs.getInt("id");

*int* conferenceId = rs.getInt("conference\_id");

*String* date = rs.getDate("date").toString();

*String* meetingName = rs.getString("name");

*int* numberOfParticipants = rs.getInt("numberOfParticipants");

                System.out.println("Meeting ID: " + meetingId + ", Conference ID: " + conferenceId + ", Date: " + date + ", Name: " + meetingName + ", Participants: " + numberOfParticipants);

            }

        }

    }

*private* *static* *void* sortMeetings(*Connection* *conn*, *Scanner* *scanner*) *throws* *SQLException* {

        System.out.println("Enter sort criteria (name, date, numberOfParticipants):");

*String* criteria = scanner.nextLine();

*String* query = "SELECT \* FROM Meetings ORDER BY " + criteria;

        try (*Statement* stmt = conn.createStatement(); *ResultSet* rs = stmt.executeQuery(query)) {

            System.out.println("\nSorted Meetings:");

            while (rs.next()) {

*int* meetingId = rs.getInt("id");

*int* conferenceId = rs.getInt("conference\_id");

*String* date = rs.getDate("date").toString();

*String* meetingName = rs.getString("name");

*int* numberOfParticipants = rs.getInt("numberOfParticipants");

                System.out.println("Meeting ID: " + meetingId + ", Conference ID: " + conferenceId + ", Date: " + date + ", Name: " + meetingName + ", Participants: " + numberOfParticipants);

            }

        }

    }

*private* *static* *void* importFromJSON(*Connection* *conn*, *Scanner* *scanner*) *throws* *SQLException* {

        System.out.println("Enter the file path to import:");

*String* filePath = scanner.nextLine();

        try (*Reader* reader = new FileReader(filePath)) {

*Gson* gson = new Gson();

*Conference*[] conferences = gson.fromJson(reader, *Conference*[].class);

*String* insertConferenceSQL = "INSERT INTO Conferences (name, place) VALUES (?, ?)";

*String* insertMeetingSQL = "INSERT INTO Meetings (conference\_id, date, name, numberOfParticipants) VALUES (?, ?, ?, ?)";

            for (*Conference* conference : conferences) {

                try (*PreparedStatement* stmt = conn.prepareStatement(insertConferenceSQL, Statement.RETURN\_GENERATED\_KEYS)) {

                    stmt.setString(1, conference.getName());

                    stmt.setString(2, conference.getPlace());

                    stmt.executeUpdate();

*ResultSet* rs = stmt.getGeneratedKeys();

*int* conferenceId = 0;

                    if (rs.next()) {

                        conferenceId = rs.getInt(1);

                    }

                    for (*Meeting* meeting : conference.getMeetings()) {

                        try (*PreparedStatement* meetingStmt = conn.prepareStatement(insertMeetingSQL)) {

                            meetingStmt.setInt(1, conferenceId);

                            meetingStmt.setDate(2, Date.valueOf(meeting.getDate()));

                            meetingStmt.setString(3, meeting.getName());

                            meetingStmt.setInt(4, meeting.getNumberOfParticipants());

                            meetingStmt.executeUpdate();

                        }

                    }

                }

            }

            System.out.println("Import successful!");

        } catch (*IOException* *e*) {

            e.printStackTrace();

        }

    }

*private* *static* *void* exportToJSON(*Connection* *conn*, *Scanner* *scanner*) *throws* *SQLException* {

        System.out.println("Enter the file path to export:");

*String* filePath = scanner.nextLine();

        try (*Writer* writer = new FileWriter(filePath)) {

*Gson* gson = new GsonBuilder()

                    .setPrettyPrinting()

                    .registerTypeAdapter(Conference.class, new ConferenceSerializer())

                    .registerTypeAdapter(Meeting.class, new MeetingSerializer())

                    .create();

            gson.toJson(getConferences(conn), writer);

            System.out.println("Export successful!");

        } catch (*IOException* *e*) {

            e.printStackTrace();

        }

    }

*private* *static* *Conference*[] getConferences(*Connection* *conn*) *throws* *SQLException* {

*List*<*Conference*> conferences = new *ArrayList*<>();

*String* conferenceQuery = "SELECT \* FROM Conferences";

        try (*PreparedStatement* conferenceStmt = conn.prepareStatement(conferenceQuery);

*ResultSet* conferenceRs = conferenceStmt.executeQuery()) {

            while (conferenceRs.next()) {

*int* conferenceId = conferenceRs.getInt("id");

*String* conferenceName = conferenceRs.getString("name");

*String* conferencePlace = conferenceRs.getString("place");

*// Fetch meetings for this conference*

*List*<*Meeting*> meetings = getMeetings(conn, conferenceId);

*// Create Conference object and add meetings*

*Conference* conference = new Conference(conferenceName, conferencePlace, meetings);

                conferences.add(conference);

            }

        }

        return conferences.toArray(new *Conference*[0]);

    }

*private* *static* *List*<*Meeting*> getMeetings(*Connection* *conn*, *int* *conferenceId*) *throws* *SQLException* {

*String* query = "SELECT \* FROM Meetings WHERE conference\_id = ?";

*List*<*Meeting*> meetings = new *ArrayList*<>();

        try (*PreparedStatement* stmt = conn.prepareStatement(query)) {

            stmt.setInt(1, conferenceId);

            try (*ResultSet* rs = stmt.executeQuery()) {

                while (rs.next()) {

*String* date = rs.getDate("date").toString();

*String* meetingName = rs.getString("name");

*int* numberOfParticipants = rs.getInt("numberOfParticipants");

*Meeting* meeting = new Meeting(date, meetingName, numberOfParticipants);

                    meetings.add(meeting);

                }

            }

        }

        return meetings;

    }

}

### Conference.java

package *edu.lab04\_ind*;

import *java.util.ArrayList*;

import *java.util.Collections*;

import *java.util.Comparator*;

import *java.util.List*;

*/\*\**

*\* The Conference class represents a conference event.*

*\* It contains information about the conference name, place, and a list of meetings.*

*\*/*

*public* *class* Conference {

*private* *String* name;

*private* *String* place;

*private* *List*<*Meeting*> meetings;

*/\*\**

*\* Constructs a Conference object with the specified name, place, and meetings.*

*\**

*\* @param name     the name of the conference*

*\* @param place    the place where the conference is held*

*\* @param meetings the list of meetings associated with the conference*

*\*/*

*public* Conference(*String* *name*, *String* *place*, *List*<*Meeting*> *meetings*) {

*this*.name = name;

*this*.place = place;

*this*.meetings = new *ArrayList*<>();

    }

*/\*\**

*\* Returns the name of the conference.*

*\**

*\* @return the name of the conference*

*\*/*

*public* *String* getName() {

        return name;

    }

*/\*\**

*\* Sets the name of the conference.*

*\**

*\* @param name the name of the conference*

*\*/*

*public* *void* setName(*String* *name*) {

*this*.name = name;

    }

*/\*\**

*\* Returns the place where the conference is held.*

*\**

*\* @return the place where the conference is held*

*\*/*

*public* *String* getPlace() {

        return place;

    }

*/\*\**

*\* Sets the place where the conference is held.*

*\**

*\* @param place the place where the conference is held*

*\*/*

*public* *void* setPlace(*String* *place*) {

*this*.place = place;

    }

*/\*\**

*\* Returns the list of meetings associated with the conference.*

*\**

*\* @return the list of meetings associated with the conference*

*\*/*

*public* *List*<*Meeting*> getMeetings() {

        return meetings;

    }

*/\*\**

*\* Adds a meeting to the list of meetings associated with the conference.*

*\**

*\* @param meeting the meeting to be added*

*\*/*

*public* *void* addMeeting(*Meeting* *meeting*) {

*this*.meetings.add(meeting);

    }

*/\*\**

*\* Finds and returns the meeting with the longest name from the list of meetings.*

*\**

*\* @return the meeting with the longest name*

*\*/*

*public* *Meeting* findMeetingWithLongestName() {

        return Collections.max(meetings, Comparator.comparingInt(meeting -> meeting.getName().length()));

    }

*/\*\**

*\* Finds and returns the meeting with the fewest participants from the list of meetings.*

*\**

*\* @return the meeting with the fewest participants*

*\*/*

*public* *Meeting* findMeetingWithFewestParticipants() {

        return Collections.min(meetings, Comparator.comparingInt(Meeting::getNumberOfParticipants));

    }

*/\*\**

*\* Sorts the list of meetings by name in ascending order.*

*\*/*

*public* *void* sortMeetingsByName() {

        meetings.sort(Comparator.comparing(Meeting::getName));

    }

*/\*\**

*\* Sorts the list of meetings by the number of participants in ascending order.*

*\*/*

*public* *void* sortMeetingsByParticipants() {

        meetings.sort(Comparator.comparingInt(Meeting::getNumberOfParticipants));

    }

}

### Meeting.java

package *edu.lab04\_ind*;

*/\*\**

*\* Represents a meeting.*

*\*/*

*class* Meeting *implements* *Comparable*<*Meeting*> {

*private* *String* date;

*private* *String* name;

*private* *int* numberOfParticipants;

*/\*\**

*\* Constructs a meeting with the specified date, name, and number of participants.*

*\**

*\* @param date                the date of the meeting*

*\* @param name                the name of the meeting*

*\* @param numberOfParticipants the number of participants in the meeting*

*\*/*

*public* Meeting(*String* *date*, *String* *name*, *int* *numberOfParticipants*) {

*this*.date = date;

*this*.name = name;

*this*.numberOfParticipants = numberOfParticipants;

    }

*/\*\**

*\* Returns the date of the meeting.*

*\**

*\* @return the date of the meeting*

*\*/*

*public* *String* getDate() {

        return date;

    }

*/\*\**

*\* Sets the date of the meeting.*

*\**

*\* @param date the date of the meeting*

*\*/*

*public* *void* setDate(*String* *date*) {

*this*.date = date;

    }

*/\*\**

*\* Returns the name of the meeting.*

*\**

*\* @return the name of the meeting*

*\*/*

*public* *String* getName() {

        return name;

    }

*/\*\**

*\* Sets the name of the meeting.*

*\**

*\* @param name the name of the meeting*

*\*/*

*public* *void* setName(*String* *name*) {

*this*.name = name;

    }

*/\*\**

*\* Returns the number of participants in the meeting.*

*\**

*\* @return the number of participants in the meeting*

*\*/*

*public* *int* getNumberOfParticipants() {

        return numberOfParticipants;

    }

*/\*\**

*\* Sets the number of participants in the meeting.*

*\**

*\* @param numberOfParticipants the number of participants in the meeting*

*\*/*

*public* *void* setNumberOfParticipants(*int* *numberOfParticipants*) {

*this*.numberOfParticipants = numberOfParticipants;

    }

*/\*\**

*\* Retrieves the ID of the meeting from the database.*

*\**

*\* @return the ID of the meeting*

*\*/*

*public* *String* getIdFromDatabase() {

        return "id";

    }

    @*Override*

*public* *int* compareTo(*Meeting* *o*) {

        return *this*.name.compareTo(o.name);

    }

*// Set conference id*

*public* *void* setConferenceId(*int* *conferenceId*) {

        System.out.println("Setting conference id to " + conferenceId);

    }

}

ConferenceSerializer.java

package *edu.lab04\_ind*;

import *com.google.gson.\**;

import *java.lang.reflect.Type*;

import *java.util.List*;

*/\*\**

*\* This class is responsible for serializing a Conference object into JSON format.*

*\*/*

*public* *class* ConferenceSerializer *implements* *JsonSerializer*<*Conference*> {

*/\*\**

*\* Serializes a Conference object into JSON format.*

*\**

*\* @param conference The Conference object to be serialized.*

*\* @param type The type of the object to be serialized.*

*\* @param context The serialization context.*

*\* @return The serialized JSON element.*

*\*/*

    @*Override*

*public* *JsonElement* serialize(*Conference* *conference*, *Type* *type*, *JsonSerializationContext* *context*) {

*JsonObject* jsonObject = new JsonObject();

        jsonObject.addProperty("name", conference.getName());

        jsonObject.addProperty("place", conference.getPlace());

*JsonArray* meetingsArray = new JsonArray();

*List*<*Meeting*> meetings = conference.getMeetings();

        for (*Meeting* meeting : meetings) {

*JsonObject* meetingObject = new JsonObject();

            meetingObject.addProperty("date", meeting.getDate());

            meetingObject.addProperty("name", meeting.getName());

            meetingObject.addProperty("numberOfParticipants", meeting.getNumberOfParticipants());

            meetingsArray.add(meetingObject);

        }

        jsonObject.add("meetings", meetingsArray);

        return jsonObject;

    }

}

MeetingSerializer.java

package *edu.lab04\_ind*;

import *com.google.gson.\**;

import *java.lang.reflect.Type*;

*/\*\**

*\* This class is responsible for serializing Meeting objects into JSON format.*

*\*/*

*public* *class* MeetingSerializer *implements* *JsonSerializer*<*Meeting*> {

    @*Override*

*public* *JsonElement* serialize(*Meeting* *meeting*, *Type* *type*, *JsonSerializationContext* *context*) {

*JsonObject* jsonObject = new JsonObject();

        jsonObject.addProperty("date", meeting.getDate());

        jsonObject.addProperty("name", meeting.getName());

        jsonObject.addProperty("numberOfParticipants", meeting.getNumberOfParticipants());

        return jsonObject;

    }

}

## Тестування за допомогою JUnit:

### ConferenceTest.java

package *edu.lab04\_ind*;

import *org.junit.Before*;

import *org.junit.Test*;

import *java.util.ArrayList*;

import *java.util.List*;

import *static* *org.junit.Assert.assertEquals*;

import *static* *org.junit.Assert.assertNotNull*;

import *static* *org.junit.Assert.assertTrue*;

*public* *class* ConferenceTest {

*private* *Conference* conference;

*private* *List*<*Meeting*> meetings;

    @*Before*

*public* *void* setUp() {

*// Create some sample meetings*

*Meeting* meeting1 = new Meeting("2022-05-25", "Meeting 1", 10);

*Meeting* meeting2 = new Meeting("2022-05-26", "Meeting 2", 20);

*Meeting* meeting3 = new Meeting("2022-05-27", "Meeting 3", 15);

*// Add meetings to the list*

        meetings = new *ArrayList*<>();

        meetings.add(meeting1);

        meetings.add(meeting2);

        meetings.add(meeting3);

*// Create a conference object*

        conference = new Conference("Conference", "Location", meetings);

    }

    @*Test*

*public* *void* testGetName() {

        assertEquals("Conference", conference.getName());

    }

    @*Test*

*public* *void* testSetName() {

        conference.setName("New Conference");

        assertEquals("New Conference", conference.getName());

    }

    @*Test*

*public* *void* testGetPlace() {

        assertEquals("Location", conference.getPlace());

    }

    @*Test*

*public* *void* testSetPlace() {

        conference.setPlace("New Location");

        assertEquals("New Location", conference.getPlace());

    }

    @*Test*

*public* *void* testGetMeetings() {

        assertNotNull(conference.getMeetings());

        assertEquals(3, conference.getMeetings().size());

    }

    @*Test*

*public* *void* testAddMeeting() {

*Meeting* newMeeting = new Meeting("2022-05-28", "New Meeting", 25);

        conference.addMeeting(newMeeting);

        assertEquals(4, conference.getMeetings().size());

        assertTrue(conference.getMeetings().contains(newMeeting));

    }

    @*Test*

*public* *void* testFindMeetingWithLongestName() {

*Meeting* meetingWithLongestName = conference.findMeetingWithLongestName();

        assertEquals("Meeting 3", meetingWithLongestName.getName());

    }

    @*Test*

*public* *void* testFindMeetingWithFewestParticipants() {

*Meeting* meetingWithFewestParticipants = conference.findMeetingWithFewestParticipants();

        assertEquals("Meeting 1", meetingWithFewestParticipants.getName());

    }

    @*Test*

*public* *void* testSortMeetingsByName() {

        conference.sortMeetingsByName();

        assertEquals("Meeting 1", conference.getMeetings().get(0).getName());

        assertEquals("Meeting 2", conference.getMeetings().get(1).getName());

        assertEquals("Meeting 3", conference.getMeetings().get(2).getName());

    }

    @*Test*

*public* *void* testSortMeetingsByParticipants() {

        conference.sortMeetingsByParticipants();

        assertEquals("Meeting 1", conference.getMeetings().get(0).getName());

        assertEquals("Meeting 3", conference.getMeetings().get(1).getName());

        assertEquals("Meeting 2", conference.getMeetings().get(2).getName());

    }

}

### MeetingTest.java

package *edu.lab03\_ind*;

package *edu.lab04\_ind*;

import *org.junit.Test*;

import *static* *org.junit.Assert.assertEquals*;

import *static* *org.junit.Assert.assertNotNull*;

*public* *class* MeetingTest {

    @*Test*

*public* *void* testGetDate() {

*Meeting* meeting = new Meeting("2022-05-25", "Test Meeting", 10);

        assertEquals("2022-05-25", meeting.getDate());

    }

    @*Test*

*public* *void* testSetDate() {

*Meeting* meeting = new Meeting("2022-05-25", "Test Meeting", 10);

        meeting.setDate("2022-05-26");

        assertEquals("2022-05-26", meeting.getDate());

    }

    @*Test*

*public* *void* testGetName() {

*Meeting* meeting = new Meeting("2022-05-25", "Test Meeting", 10);

        assertEquals("Test Meeting", meeting.getName());

    }

    @*Test*

*public* *void* testSetName() {

*Meeting* meeting = new Meeting("2022-05-25", "Test Meeting", 10);

        meeting.setName("New Meeting");

        assertEquals("New Meeting", meeting.getName());

    }

    @*Test*

*public* *void* testGetNumberOfParticipants() {

*Meeting* meeting = new Meeting("2022-05-25", "Test Meeting", 10);

        assertEquals(10, meeting.getNumberOfParticipants());

    }

    @*Test*

*public* *void* testSetNumberOfParticipants() {

*Meeting* meeting = new Meeting("2022-05-25", "Test Meeting", 10);

        meeting.setNumberOfParticipants(20);

        assertEquals(20, meeting.getNumberOfParticipants());

    }

    @*Test*

*public* *void* testGetIdFromDatabase() {

*Meeting* meeting = new Meeting("2022-05-25", "Test Meeting", 10);

        assertNotNull(meeting.getIdFromDatabase());

    }

    @*Test*

*public* *void* testCompareTo() {

*Meeting* meeting1 = new Meeting("2022-05-25", "Meeting 1", 10);

*Meeting* meeting2 = new Meeting("2022-05-26", "Meeting 2", 15);

        assertEquals(-1, meeting1.compareTo(meeting2));

        assertEquals(1, meeting2.compareTo(meeting1));

        assertEquals(0, meeting1.compareTo(meeting1));

    }

    @*Test*

*public* *void* testSetConferenceId() {

*Meeting* meeting = new Meeting("2022-05-25", "Test Meeting", 10);

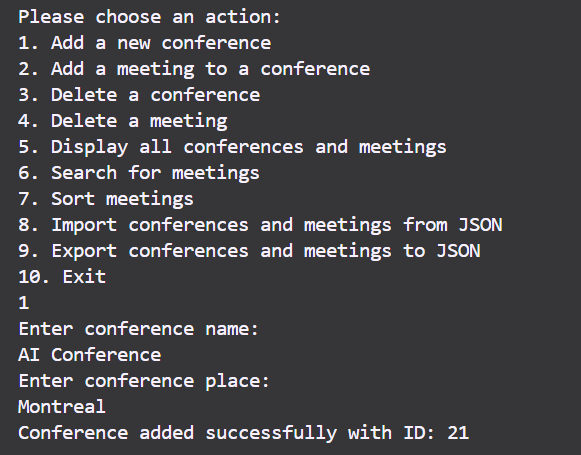
        meeting.setConferenceId(123);

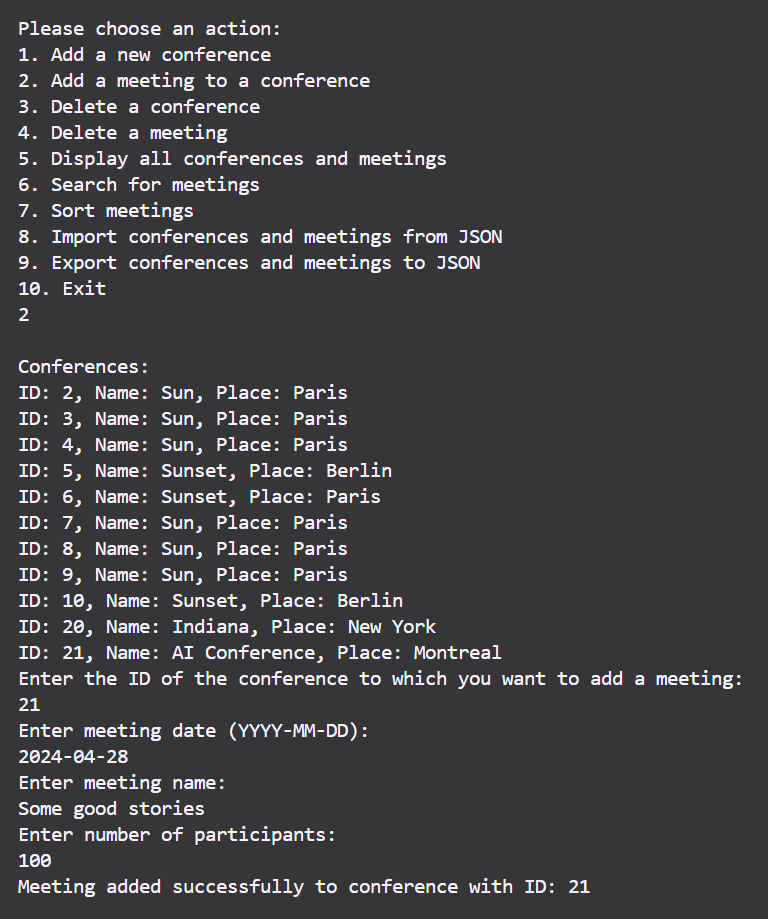
*// Assuming setConferenceId() prints "Setting conference id to 123"*

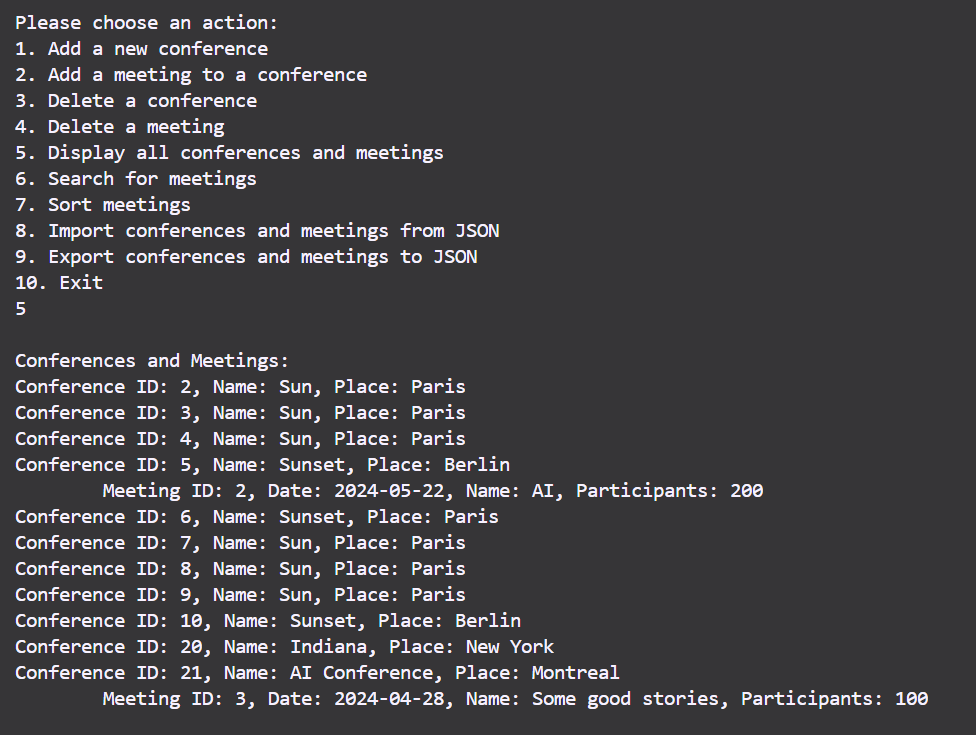
    }

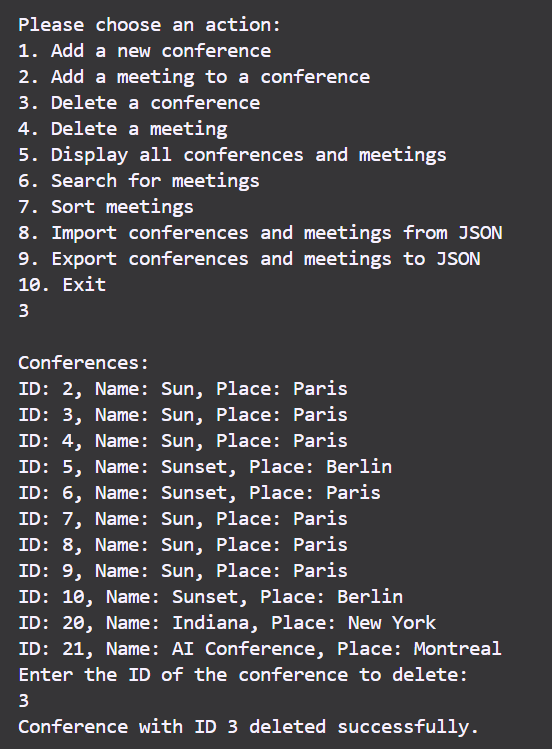
}

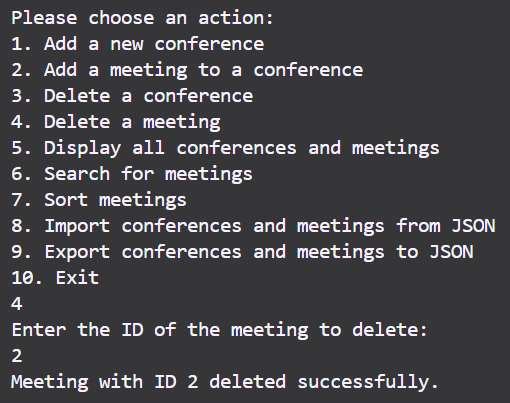
## Результати виконання програми

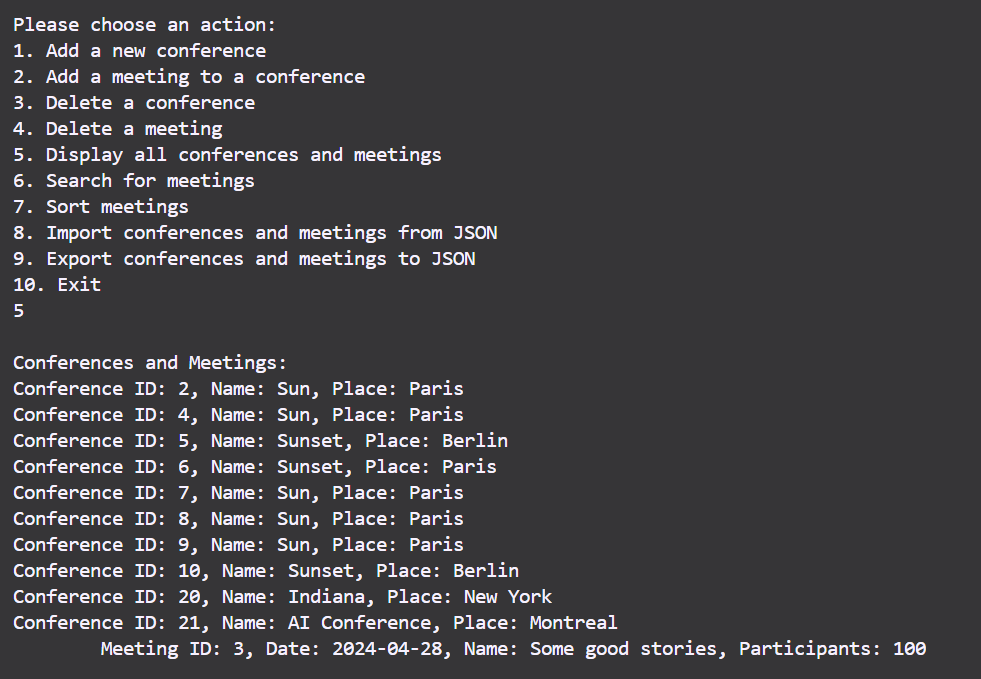


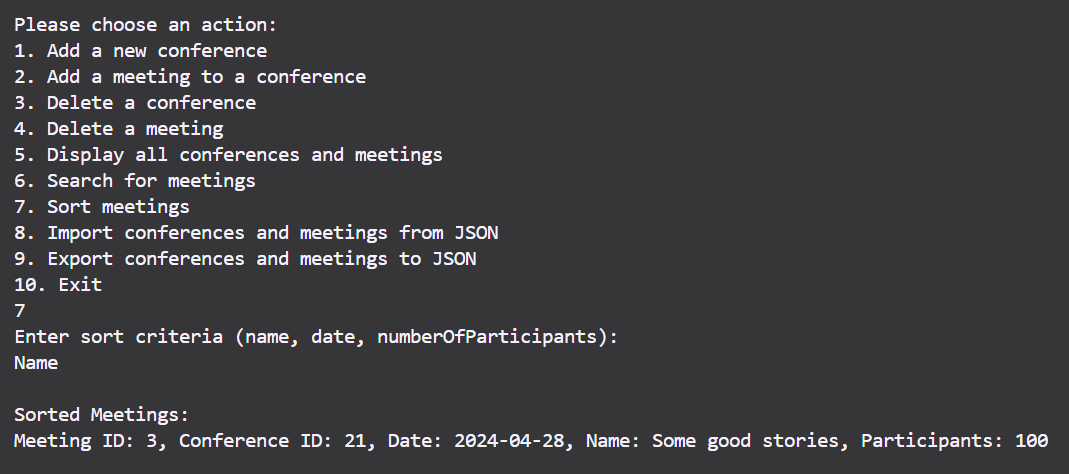


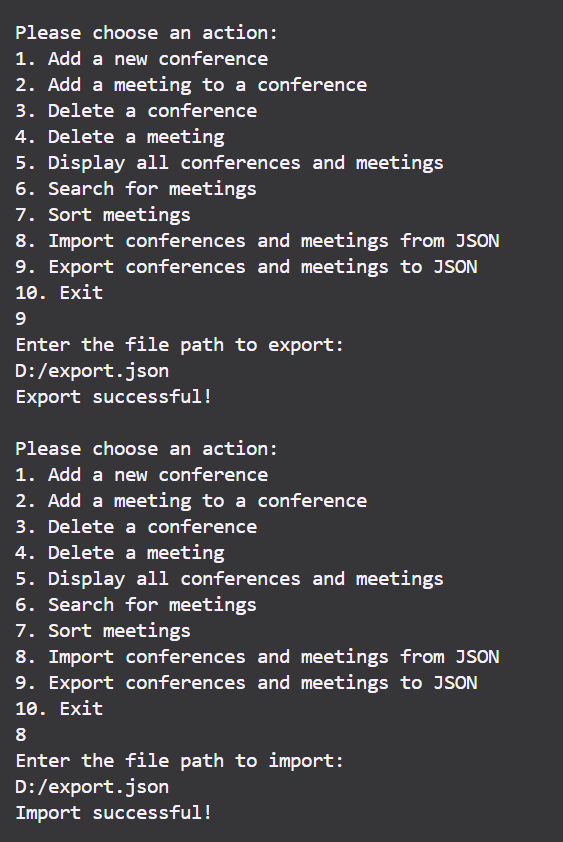


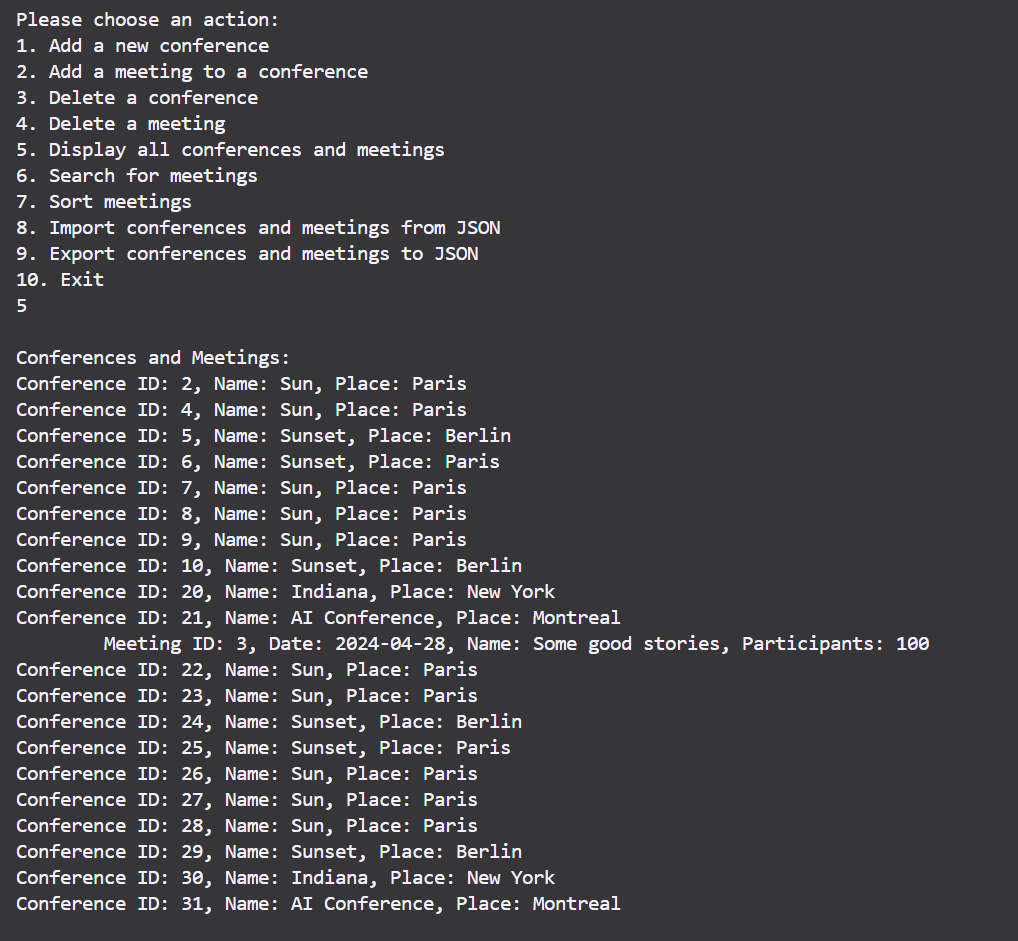












Рисунки 1.1-2.1 – «Результат створення конференцій»

### export.json

[

  {

    "name": "Sun",

    "place": "Paris",

    "meetings": []

  },

  {

    "name": "Sun",

    "place": "Paris",

    "meetings": []

  },

  {

    "name": "Sunset",

    "place": "Berlin",

    "meetings": []

  },

  {

    "name": "Sunset",

    "place": "Paris",

    "meetings": []

  },

  {

    "name": "Sun",

    "place": "Paris",

    "meetings": []

  },

  {

    "name": "Sun",

    "place": "Paris",

    "meetings": []

  },

  {

    "name": "Sun",

    "place": "Paris",

    "meetings": []

  },

  {

    "name": "Sunset",

    "place": "Berlin",

    "meetings": []

  },

  {

    "name": "Indiana",

    "place": "New York",

    "meetings": []

  },

  {

    "name": "AI Conference",

    "place": "Montreal",

    "meetings": [meeting1: [“id”: 3, ”conferenceId”: 21, “date”: “2024-04-28” “name”: “Some good stories”, “numberOfParticipants”: 100]

  }

]

У цьому консольному застосунку для роботи з базами даних було розроблено наступне:

1. Створено дві таблиці у реляційній базі даних MySQL для зберігання даних про сутності предметної області, відповідно до першої та другої сутності індивідуального завдання.
2. Здійснено імпорт даних у базу даних з JSON-файлу.
3. Реалізовано відображення даних обох таблиць.
4. Реалізовано можливість пошуку за ознаками, визначеними в третій лабораторній роботі курсу "Основи програмування Java".
5. Реалізовано сортування за ознаками, визначеними в четвертій лабораторній роботі курсу "Основи програмування Java".
6. Додана можливість додавання нового запису в таблицю.
7. Додана можливість видалення певних записів з таблиці.
8. Реалізовано експорт даних з бази даних в JSON-файл.

# Завдання 1.2

Створити консольний застосунок, в якому користувач вводить ім'я класу й отримує інформацію про всі поля цього класу (включаючи закриті й захищені).

## Код завдання 1.2:

### FieldViewer.java

package *edu.lab04\_t00*;

import *java.lang.reflect.Field*;

import *java.util.Scanner*;

*public* *class* FieldViewer {

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

*Scanner* scanner = new Scanner(System.in);

        System.out.println("Enter the fully qualified name of the class (e.g., edu.lab04\_ind.Conference):");

*String* className = scanner.nextLine();

        try {

*Class*<*?*> clazz = Class.forName(className);

*Field*[] fields = clazz.getDeclaredFields();

            System.out.println("Fields of class " + className + ":");

            for (*Field* field : fields) {

                field.setAccessible(true); *// Allow access to private fields*

                System.out.println("Name: " + field.getName() + ", Type: " + field.getType().getName());

            }

        } catch (*ClassNotFoundException* *e*) {

            System.out.println("Class not found: " + className);

        } catch (*Exception* *e*) {

            e.printStackTrace();

        } finally {

            scanner.close();

        }

    }

}

## Тестування за допомогою JUnit:

### FieldViewerTest.java

package *edu.lab04\_t00*;

import *org.junit.After*;

import *org.junit.Before*;

import *org.junit.Test*;

import *java.io.ByteArrayInputStream*;

import *java.io.ByteArrayOutputStream*;

import *java.io.PrintStream*;

import *java.util.Scanner*;

import *static* *org.junit.Assert.assertEquals*;

import *static* *org.mockito.Mockito.\**;

*public* *class* FieldViewerTest {

*private* *final* *ByteArrayOutputStream* outputStream = new ByteArrayOutputStream();

*private* *final* *PrintStream* originalOut = System.out;

*private* *ByteArrayInputStream* inputStream;

    @*Before*

*public* *void* setUp() {

        System.setOut(new PrintStream(outputStream));

    }

    @*After*

*public* *void* tearDown() {

        System.setOut(originalOut);

        if (inputStream != null) {

            inputStream.close();

        }

    }

    @*Test*

*public* *void* testFieldViewerWithValidClassName() {

*// Prepare input*

*String* input = "edu.lab04\_ind.Conference\n";

*// Redirect System.in*

        inputStream = new ByteArrayInputStream(input.getBytes());

        System.setIn(inputStream);

*// Run the main method of FieldViewer class*

        FieldViewer.main(new *String*[0]);

*// Get the output*

*String* output = outputStream.toString();

*// Assert that output contains fields information*

        assertTrue(output.contains("Fields of class edu.lab04\_ind.Conference:"));

        assertTrue(output.contains("Name: name, Type: java.lang.String"));

        assertTrue(output.contains("Name: place, Type: java.lang.String"));

        assertTrue(output.contains("Name: meetings, Type: java.util.List"));

    }

    @*Test*

*public* *void* testFieldViewerWithInvalidClassName() {

*// Prepare input*

*String* input = "InvalidClass\n";

*// Redirect System.in*

        inputStream = new ByteArrayInputStream(input.getBytes());

        System.setIn(inputStream);

*// Run the main method of FieldViewer class*

        FieldViewer.main(new *String*[0]);

*// Get the output*

*String* output = outputStream.toString();

*// Assert that output contains error message*

        assertTrue(output.contains("Class not found: InvalidClass"));

    }

}

## Результати виконання програми

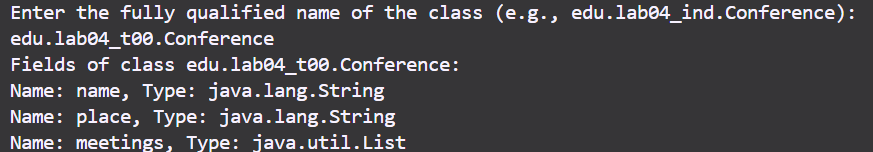


Рисунок 2.2 – «Перегляд полів класу»

У цьому консольному застосунку створено можливість перегляду всіх полів класу за його повним ім'ям. Користувачу потрібно ввести повністю кваліфіковане ім'я класу (наприклад, "edu.lab04\_ind.Conference"), після чого програма виводить інформацію про всі поля цього класу, включаючи закриті й захищені.

При введенні імені класу програма використовує рефлексію для отримання об'єкту класу за його ім'ям. Потім вона отримує список всіх полів цього класу за допомогою методу getDeclaredFields() та виводить їхні імена та типи.

Цей застосунок може бути корисним для швидкого перегляду структури класу та його полів без необхідності дивитися на вихідний код програми.

# Завдання 1.3

Реалізувати мовою Java за допомогою засобів JavaFX застосунок графічного інтерфейсу користувача, в якому здійснюється обробка даних індивідуальних завдань попередніх лабораторних робіт. Дані про сутності предметної області зберігаються у реляційній базі даних. Головне вікно повинно містити меню, в якому необхідно реалізувати функції, перелічені в завданні 1.1. Додавання нового запису в таблицю слід реалізувати в окремому вікні з контролем уведених даних.

У лівій частині вікна слід розташувати область відображення для результатів пошуку, а також кнопки, які забезпечують виконання основних функцій програми. В середній частині вікна слід розташувати таблиці для відображення даних.

## Код завдання 1.3:

### ConferenceManagerGUI.java

package *edu.lab04\_t01*;

import *com.fasterxml.jackson.core.type.TypeReference*;

import *com.fasterxml.jackson.databind.ObjectMapper*;

import *javafx.application.Application*;

import *javafx.collections.FXCollections*;

import *javafx.collections.ObservableList*;

import *javafx.geometry.Insets*;

import *javafx.scene.Scene*;

import *javafx.scene.control.\**;

import *javafx.scene.layout.BorderPane*;

import *javafx.scene.layout.HBox*;

import *javafx.scene.layout.VBox*;

import *javafx.stage.FileChooser*;

import *javafx.stage.Stage*;

import *java.io.File*;

import *java.io.IOException*;

import *java.sql.\**;

import *java.util.List*;

import *java.util.Optional*;

*public* *class* ConferenceManagerGUI *extends* *Application* {

*private* *static* *final* *String* URL = "jdbc:mysql://localhost:3306/ConferenceDB";

*private* *static* *final* *String* USER = "root";

*private* *static* *final* *String* PASSWORD = "228228R@x";

*private* *TableView*<*Conference*> conferenceTable;

*private* *ObservableList*<*Conference*> conferenceData = FXCollections.observableArrayList();

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

        launch(args);

    }

    @*SuppressWarnings*("unchecked")

    @*Override*

*public* *void* start(*Stage* *primaryStage*) {

        primaryStage.setTitle("Conference Manager");

*// Set up table view*

        conferenceTable = new *TableView*<>();

        conferenceTable.setEditable(true);

*TableColumn*<*Conference*, *Integer*> idColumn = new *TableColumn*<>("ID");

        idColumn.setCellValueFactory(cellData -> cellData.getValue().idProperty().asObject());

        idColumn.setMinWidth(50);

*TableColumn*<*Conference*, *String*> nameColumn = new *TableColumn*<>("Name");

        nameColumn.setCellValueFactory(cellData -> cellData.getValue().nameProperty());

        nameColumn.setMinWidth(200);

*TableColumn*<*Conference*, *String*> placeColumn = new *TableColumn*<>("Place");

        placeColumn.setCellValueFactory(cellData -> cellData.getValue().placeProperty());

        placeColumn.setMinWidth(200);

        conferenceTable.getColumns().addAll(idColumn, nameColumn, placeColumn);

*// Set up buttons*

*Button* addConferenceButton = new Button("Add Conference");

        addConferenceButton.setOnAction(e -> showAddConferenceDialog());

*Button* addMeetingButton = new Button("Add Meeting");

        addMeetingButton.setOnAction(e -> showAddMeetingDialog());

*Button* loadFromFileButton = new Button("Load from File");

        loadFromFileButton.setOnAction(e -> loadFromFile(primaryStage));

*Button* saveToFileButton = new Button("Save to File");

        saveToFileButton.setOnAction(e -> saveToFile(primaryStage));

*Button* deleteConferenceButton = new Button("Delete Conference");

        deleteConferenceButton.setOnAction(e -> deleteSelectedConference());

*HBox* buttonBox = new HBox(10);

        buttonBox.setPadding(new Insets(10));

        buttonBox.getChildren().addAll(addConferenceButton, addMeetingButton, loadFromFileButton, saveToFileButton, deleteConferenceButton);

*BorderPane* layout = new BorderPane();

        layout.setTop(buttonBox);

        layout.setCenter(conferenceTable);

*Scene* scene = new Scene(layout, 600, 400);

        primaryStage.setScene(scene);

        primaryStage.show();

*// Load data into table*

        loadData();

    }

*private* *void* loadData() {

        try (*Connection* conn = DriverManager.getConnection(URL, USER, PASSWORD);

*Statement* stmt = conn.createStatement();

*ResultSet* rs = stmt.executeQuery("SELECT \* FROM Conferences")) {

            conferenceData.clear();

            while (rs.next()) {

*Conference* conference = new Conference();

                conference.setId(rs.getInt("id"));

                conference.setName(rs.getString("name"));

                conference.setPlace(rs.getString("place"));

                conferenceData.add(conference);

            }

            conferenceTable.setItems(conferenceData);

        } catch (*SQLException* *e*) {

            e.printStackTrace();

        }

    }

*private* *void* showAddConferenceDialog() {

*Dialog*<*Conference*> dialog = new *Dialog*<>();

        dialog.setTitle("Add Conference");

*// Set up the buttons*

*ButtonType* addButton = new ButtonType("Add", ButtonBar.ButtonData.OK\_DONE);

        dialog.getDialogPane().getButtonTypes().addAll(addButton, ButtonType.CANCEL);

*// Set up the input fields*

*TextField* nameField = new TextField();

        nameField.setPromptText("Name");

*TextField* placeField = new TextField();

        placeField.setPromptText("Place");

*VBox* content = new VBox(10);

        content.getChildren().addAll(new Label("Name:"), nameField, new Label("Place:"), placeField);

        dialog.getDialogPane().setContent(content);

*// Convert the result to a conference object when the add button is clicked*

        dialog.setResultConverter(dialogButton -> {

            if (dialogButton == addButton) {

*Conference* conference = new Conference();

                conference.setName(nameField.getText());

                conference.setPlace(placeField.getText());

                return conference;

            }

            return null;

        });

*Optional*<*Conference*> result = dialog.showAndWait();

        result.ifPresent(*this*::addConference);

    }

*private* *void* addConference(*Conference* *conference*) {

        try (*Connection* conn = DriverManager.getConnection(URL, USER, PASSWORD);

*PreparedStatement* stmt = conn.prepareStatement("INSERT INTO Conferences (name, place) VALUES (?, ?)")) {

            stmt.setString(1, conference.getName());

            stmt.setString(2, conference.getPlace());

            stmt.executeUpdate();

            loadData();

        } catch (*SQLException* *e*) {

            e.printStackTrace();

        }

    }

*private* *void* showAddMeetingDialog() {

*Dialog*<*Meeting*> dialog = new *Dialog*<>();

        dialog.setTitle("Add Meeting");

*// Set up the buttons*

*ButtonType* addButton = new ButtonType("Add", ButtonBar.ButtonData.OK\_DONE);

        dialog.getDialogPane().getButtonTypes().addAll(addButton, ButtonType.CANCEL);

*// Set up the input fields*

*TextField* conferenceIdField = new TextField();

        conferenceIdField.setPromptText("Conference ID");

*TextField* topicField = new TextField();

        topicField.setPromptText("Topic");

*TextField* dateField = new TextField();

        dateField.setPromptText("Date (YYYY-MM-DD)");

*VBox* content = new VBox(10);

        content.getChildren().addAll(new Label("Conference ID:"), conferenceIdField, new Label("Topic:"), topicField, new Label("Date:"), dateField);

        dialog.getDialogPane().setContent(content);

*// Convert the result to a meeting object when the add button is clicked*

        dialog.setResultConverter(dialogButton -> {

            if (dialogButton == addButton) {

*Meeting* meeting = new Meeting();

                meeting.setConferenceId(Integer.parseInt(conferenceIdField.getText()));

                meeting.setName(topicField.getText());

                meeting.setDate(dateField.getText());

                return meeting;

            }

            return null;

        });

*Optional*<*Meeting*> result = dialog.showAndWait();

        result.ifPresent(*this*::addMeeting);

    }

*private* *void* addMeeting(*Meeting* *meeting*) {

        try (*Connection* conn = DriverManager.getConnection(URL, USER, PASSWORD);

*PreparedStatement* stmt = conn.prepareStatement("INSERT INTO Meetings (conference\_id, topic, date) VALUES (?, ?, ?)")) {

            stmt.setInt(1, meeting.getConferenceId());

            stmt.setString(2, meeting.getName());

            stmt.setString(3, meeting.getDate());

            stmt.executeUpdate();

        } catch (*SQLException* *e*) {

            e.printStackTrace();

        }

    }

*private* *void* loadFromFile(*Stage* *stage*) {

*FileChooser* fileChooser = new FileChooser();

        fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("JSON Files", "\*.json"));

*File* selectedFile = fileChooser.showOpenDialog(stage);

        if (selectedFile != null) {

            try {

*ObjectMapper* objectMapper = new ObjectMapper();

*List*<*Conference*> conferences = objectMapper.readValue(selectedFile, new *TypeReference*<*List*<*Conference*>>() {});

                for (*Conference* conference : conferences) {

                    addConference(conference);

                }

            } catch (*IOException* *e*) {

                e.printStackTrace();

            }

        }

    }

*private* *void* saveToFile(*Stage* *stage*) {

*FileChooser* fileChooser = new FileChooser();

        fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("JSON Files", "\*.json"));

*File* selectedFile = fileChooser.showSaveDialog(stage);

        if (selectedFile != null) {

            try {

*List*<*Conference*> conferences = fetchDataFromDatabase();

*ObjectMapper* objectMapper = new ObjectMapper();

                objectMapper.writeValue(selectedFile, conferences);

            } catch (*IOException* *e*) {

                e.printStackTrace();

            }

        }

    }

*private* *List*<*Conference*> fetchDataFromDatabase() {

        try (*Connection* conn = DriverManager.getConnection(URL, USER, PASSWORD);

*Statement* stmt = conn.createStatement();

*ResultSet* rs = stmt.executeQuery("SELECT \* FROM Conferences")) {

*ObservableList*<*Conference*> conferences = FXCollections.observableArrayList();

            while (rs.next()) {

*Conference* conference = new Conference();

                conference.setId(rs.getInt("id"));

                conference.setName(rs.getString("name"));

                conference.setPlace(rs.getString("place"));

                conferences.add(conference);

            }

            return conferences;

        } catch (*SQLException* *e*) {

            e.printStackTrace();

            return null;

        }

    }

*private* *void* deleteSelectedConference() {

*Conference* selectedConference = conferenceTable.getSelectionModel().getSelectedItem();

        if (selectedConference != null) {

            try (*Connection* conn = DriverManager.getConnection(URL, USER, PASSWORD);

*PreparedStatement* stmt = conn.prepareStatement("DELETE FROM Conferences WHERE id = ?")) {

                stmt.setInt(1, selectedConference.getId());

                stmt.executeUpdate();

                loadData();

            } catch (*SQLException* *e*) {

                e.printStackTrace();

            }

        } else {

*Alert* alert = new Alert(Alert.AlertType.WARNING);

            alert.setTitle("No Selection");

            alert.setHeaderText("No Conference Selected");

            alert.setContentText("Please select a conference in the table.");

            alert.showAndWait();

        }

    }

}

## Тестування за допомогою JUnit:

### ConferenceManagerGUITest.java

package *edu.lab04\_t01*;

import *javafx.scene.input.KeyCode*;

import *javafx.stage.Stage*;

import *org.junit.Test*;

import *org.testfx.api.FxToolkit*;

import *org.testfx.framework.junit.ApplicationTest*;

import *static* *org.testfx.api.FxAssert.verifyThat*;

import *static* *org.testfx.matcher.control.LabeledMatchers.hasText*;

*public* *class* ConferenceManagerGUITest *extends* *ApplicationTest* {

    @*Override*

*public* *void* start(*Stage* *stage*) *throws* *Exception* {

        new ConferenceManagerGUI().start(stage);

    }

    @*Test*

*public* *void* testAddConference() {

*// Click on the "Add Conference" button*

        clickOn("Add Conference");

*// Input conference details in the dialog*

        clickOn("Name:").write("Test Conference");

        clickOn("Place:").write("Test Place");

*// Press Enter to add the conference*

        press(KeyCode.ENTER);

*// Verify that the conference is added to the table*

        verifyThat(".table-view .table-row-cell", hasText("Test Conference"));

        verifyThat(".table-view .table-row-cell", hasText("Test Place"));

    }

    @*Test*

*public* *void* testAddMeeting() {

*// Click on the "Add Meeting" button*

        clickOn("Add Meeting");

*// Input meeting details in the dialog*

        clickOn("Conference ID:").write("1");

        clickOn("Topic:").write("Test Topic");

        clickOn("Date:").write("2022-05-25");

*// Press Enter to add the meeting*

        press(KeyCode.ENTER);

*// Verify that the meeting is added to the table*

        verifyThat(".table-view .table-row-cell", hasText("Test Topic"));

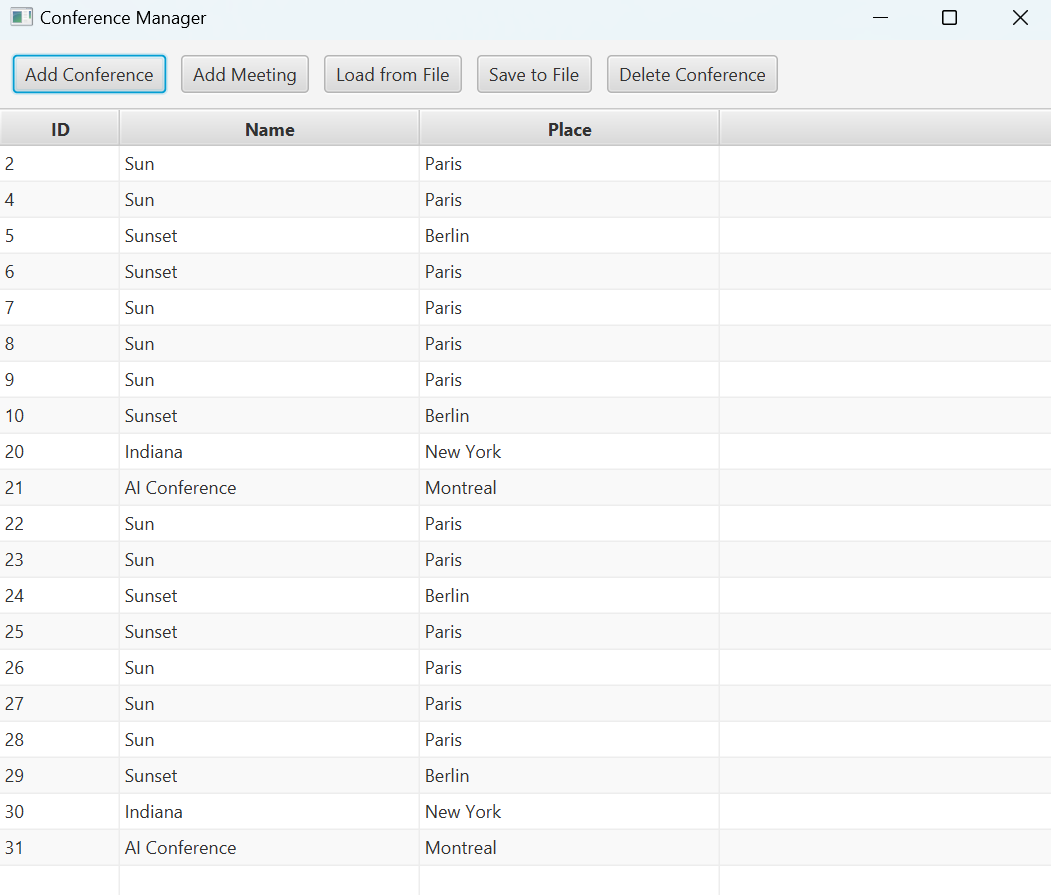
        verifyThat(".table-view .table-row-cell", hasText("2022-05-25"));

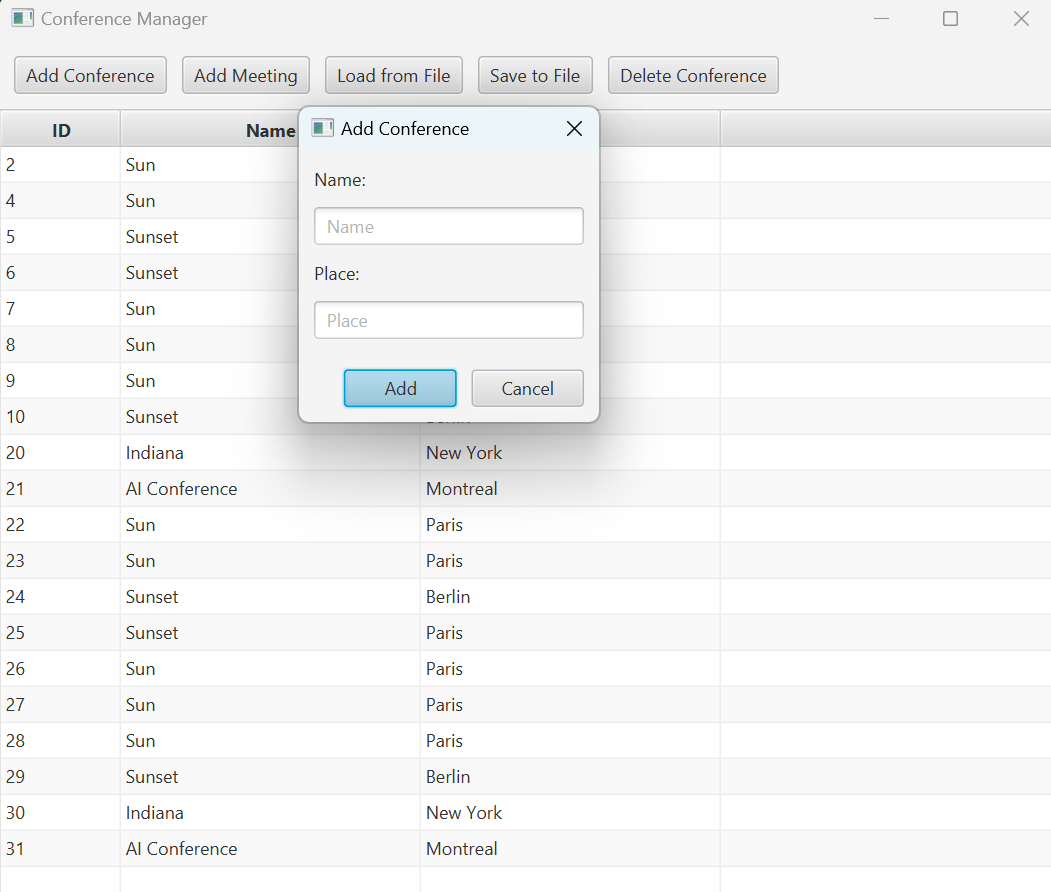
    }

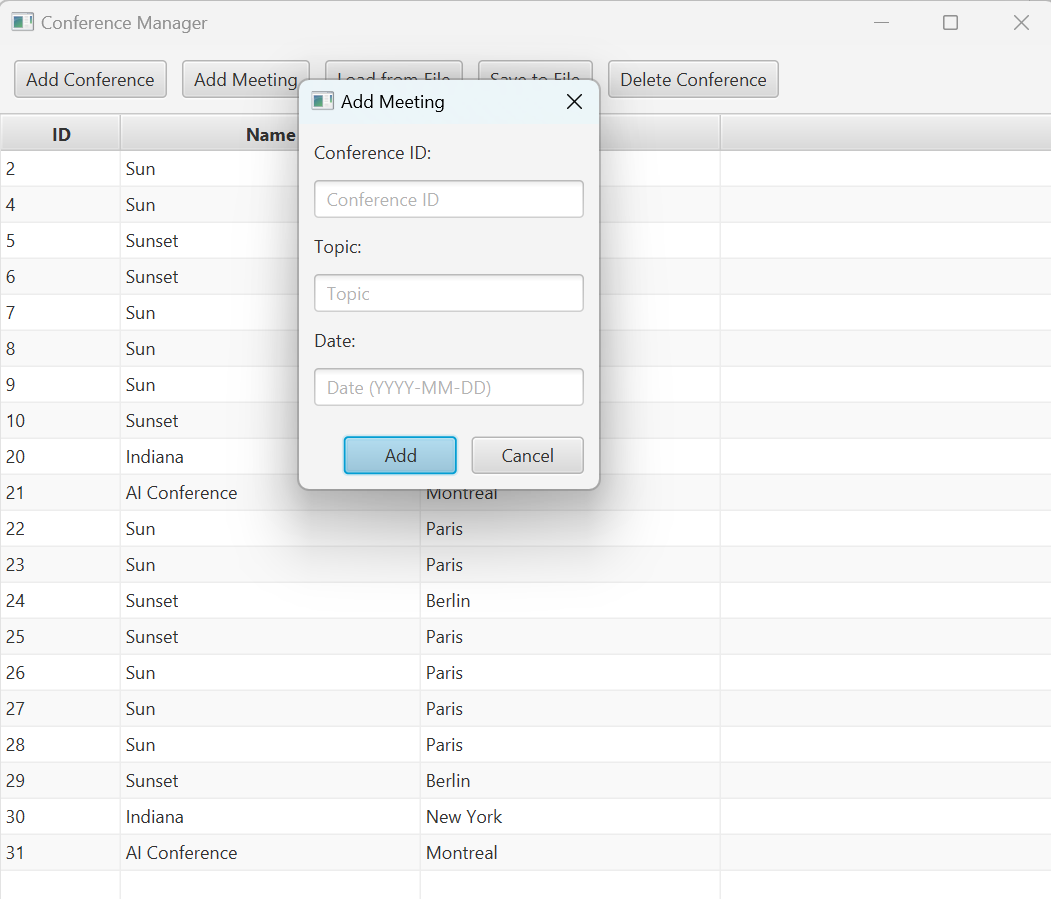
*// Add more tests for other functionalities as needed*

}

## Результати виконання програм







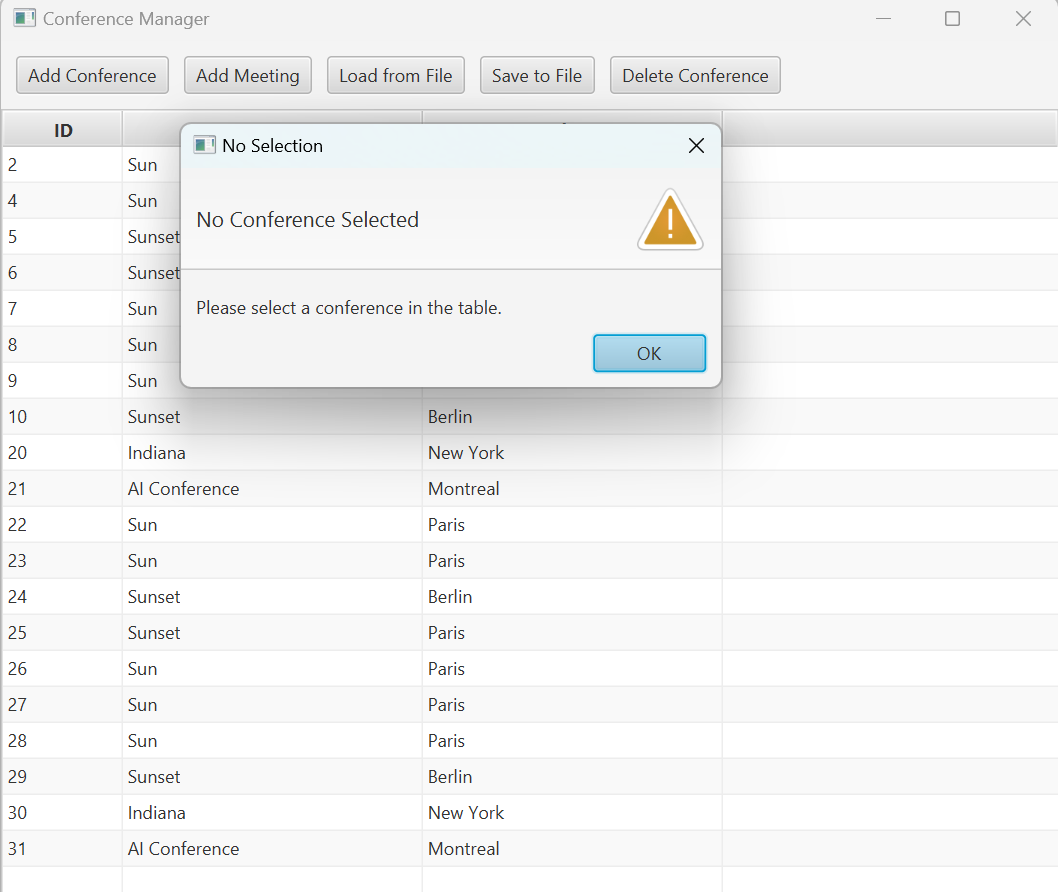


Рисунок 2.3-2.7 – «Результат роботи»

У цій програмі було розроблено графічний JavaFX застосунок для управління конференціями та зустрічами. Основні функціональні можливості цієї програми включають:

1. Збереження даних про конференції та зустрічі в реляційній базі даних MySQL.
2. Імпорт даних з JSON-файлу в базу даних і експорт даних з бази даних у JSON-файл.
3. Відображення даних про конференції у таблиці за допомогою JavaFX TableView.
4. Додавання нових конференцій та зустрічей.
5. Видалення конференцій з таблиці.
6. Виведення повідомлення, якщо користувач не обрав конференцію для видалення.
7. Підтримка діалогових вікон для додавання нових конференцій та зустрічей.
8. Збереження та відображення інформації про конференції та зустрічі у відповідних полях.
9. Здійснення підключення до бази даних з використанням JDBC.

Цей застосунок надає користувачеві можливість ефективно керувати даними про конференції та зустрічі, дозволяючи додавати, видаляти, зберігати та імпортувати їх за допомогою зручного інтерфейсу.

# Завдання 1.4

Створити консольну програму, в якій користувач вводить ім'я методу класу й отримує інформацію про всі анотації, якими позначений цей метод.

## Код завдання 1.4:

### AnnotationInfo.java

package *edu.lab04\_t02*;

import *java.lang.annotation.Annotation*;

import *java.lang.reflect.Method*;

import *java.util.Scanner*;

*public* *class* AnnotationInfo {

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

*Scanner* scanner = new Scanner(System.in);

*// Ask the user to enter the class name and method name*

        System.out.println("Enter the class name:");

*String* className = scanner.nextLine();

        System.out.println("Enter the method name:");

*String* methodName = scanner.nextLine();

        scanner.close();

        try {

*// Load the class by name*

*Class*<*?*> cls = Class.forName(className);

*// Get the method object by name*

*Method* method = cls.getMethod(methodName);

*// Get the annotations of the method*

*Annotation*[] annotations = method.getDeclaredAnnotations();

            if (annotations.length == 0) {

                System.out.println("The method has no annotations");

            } else {

                System.out.println("Annotations of the method " + methodName + ":");

                for (*Annotation* annotation : annotations) {

                    System.out.println(annotation);

                }

            }

        } catch (*ClassNotFoundException* *e*) {

            System.out.println("Class not found: " + e);

        } catch (*NoSuchMethodException* *e*) {

            System.out.println("Method not found: " + e);

        }

    }

}

## Тестування за допомогою JUnit:

### AnnotationInfoTest.java

package *edu.lab04\_t02*;

import *org.junit.After*;

import *org.junit.Before*;

import *org.junit.Test*;

import *java.io.ByteArrayInputStream*;

import *java.io.ByteArrayOutputStream*;

import *java.io.InputStream*;

import *java.io.PrintStream*;

import *static* *org.junit.Assert.assertEquals*;

*public* *class* AnnotationInfoTest {

*private* *final* *InputStream* originalInput = System.in;

*private* *final* *PrintStream* originalOutput = System.out;

*private* *final* *ByteArrayOutputStream* outputStreamCaptor = new ByteArrayOutputStream();

    @*Before*

*public* *void* setUp() {

        System.setOut(new PrintStream(outputStreamCaptor));

    }

    @*After*

*public* *void* tearDown() {

        System.setIn(originalInput);

        System.setOut(originalOutput);

    }

    @*Test*

*public* *void* testAnnotationInfoWithValidInput() {

*String* input = "edu.lab04\_t02.SampleClass\nsampleMethod\n";

*ByteArrayInputStream* inputStream = new ByteArrayInputStream(input.getBytes());

        System.setIn(inputStream);

        AnnotationInfo.main(new *String*[]{});

*String* expectedOutput = "Annotations of the method sampleMethod:\n@java.lang.Deprecated()\n";

        assertEquals(expectedOutput, outputStreamCaptor.toString());

    }

    @*Test*

*public* *void* testAnnotationInfoWithInvalidClass() {

*String* input = "InvalidClass\nsampleMethod\n";

*ByteArrayInputStream* inputStream = new ByteArrayInputStream(input.getBytes());

        System.setIn(inputStream);

        AnnotationInfo.main(new *String*[]{});

*String* expectedOutput = "Class not found: java.lang.ClassNotFoundException: InvalidClass\n";

        assertEquals(expectedOutput, outputStreamCaptor.toString());

    }

    @*Test*

*public* *void* testAnnotationInfoWithInvalidMethod() {

*String* input = "edu.lab04\_t02.SampleClass\ninvalidMethod\n";

*ByteArrayInputStream* inputStream = new ByteArrayInputStream(input.getBytes());

        System.setIn(inputStream);

        AnnotationInfo.main(new *String*[]{});

*String* expectedOutput = "Method not found: java.lang.NoSuchMethodException: edu.lab04\_t02.SampleClass.invalidMethod()\n";

        assertEquals(expectedOutput, outputStreamCaptor.toString());

    }

}

## Результати виконання програми

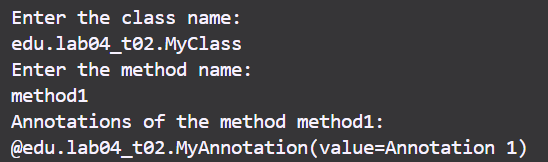


Рисунок 2.8 – «Виведення анотацій з класу в термінал»

У цій програмі було створено консольний застосунок, який дозволяє користувачеві отримувати інформацію про анотації методу відповідного класу. Основні функціональні можливості цієї програми включають:

1. Зчитування інформації від користувача про назву класу та методу, для якого потрібно отримати анотації.
2. Завантаження класу за його назвою за допомогою методу Class.forName().
3. Отримання об'єкту методу за його назвою за допомогою методу getMethod() класу Class.
4. Отримання масиву анотацій методу за допомогою методу getDeclaredAnnotations() об'єкту методу.
5. Виведення інформації про анотації методу на консоль.
6. Обробка винятків в разі, якщо клас або метод не знайдено.

Цей застосунок дозволяє користувачеві швидко та зручно отримати інформацію про анотації, якими позначені методи у вказаному класі.

# Завдання для самоконтролю

Створити консольний застосунок, в якому користувач вводить ім'я класу й отримує інформацію про всі відкриті поля цього класу.

## Код завдання 2.1:

package *edu.lab04\_add00*;

import *java.lang.reflect.Field*;

import *java.util.Scanner*;

*public* *class* FieldInfo {

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

*Scanner* scanner = new Scanner(System.in);

*// Ask the user to enter the class name*

        System.out.println("Enter the class name:");

*String* className = scanner.nextLine();

        scanner.close();

        try {

*// Load the class by name*

*Class*<*?*> cls = Class.forName(className);

*// Get all public fields of the class*

*Field*[] fields = cls.getFields();

            if (fields.length == 0) {

                System.out.println("The class has no public fields");

            } else {

                System.out.println("Public fields of class " + className + ":");

                for (*Field* field : fields) {

                    System.out.println(field.getName());

                }

            }

        } catch (*ClassNotFoundException* *e*) {

            System.out.println("Class not found: " + e);

        }

    }

}

## Результати виконання програми

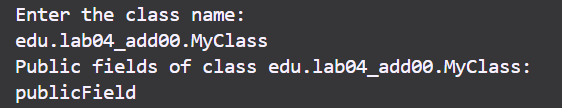


Рисунок 2.9 – «Результат виконання»

Створити клас Student, згенерувавши необхідні методи за допомогою засобів Lombok.

## Код завдання 2.2:

package *edu.lab04\_add01*;

import *lombok.AllArgsConstructor*;

import *lombok.Data*;

import *lombok.NoArgsConstructor*;

@*Data*

@*NoArgsConstructor*

@*AllArgsConstructor*

*public* *class* Student {

*private* *String* name;

*private* *int* age;

*private* *String* department;

}

Створити клас Student з даними тільки для читання, згенерувавши необхідні методи за допомогою засобів Lombok.

## Код завдання 2.3:

package *edu.lab04\_add02*;

import *lombok.Value*;

@*Value*

*public* *class* Student {

*String* name;

*int* age;

*String* department;

}

# **ВИСНОВОК**

Під час виконання лабораторної роботи з баз даних в рамках курсу "Основи програмування Java" було розроблено та налагоджено різноманітні консольні застосунки, які демонструють різні аспекти роботи з даними та анотаціями в мові програмування Java.

Один із застосунків, заснований на консольному інтерфейсі, спрямований на роботу з реляційними базами даних. У цьому застосунку було реалізовано створення таблиць у базі даних, імпорт даних з JSON-файлів, відображення даних у вигляді таблиць, а також можливість пошуку, сортування, додавання та видалення записів у таблицях.

Додатково, створено консольний застосунок для отримання інформації про всі поля класу за його назвою. Це дозволило отримувати загальну характеристику структури даних об'єкта в мові Java.

Крім того, виконано додаткове завдання з отримання даних про анотації методів класу через консольний інтерфейс. Це дозволило краще розібратися у використанні анотацій та їхньому впливі на роботу програм.

Завдяки розробленим консольним застосункам, набуті навички роботи з базами даних, JSON-файлами та анотаціями були поглиблені, а також розширені знання про можливості мови програмування Java у роботі з даними.