**Міністерство освіти і науки України**

**Національний технічний університет**

**«Дніпровська політехніка»**



Лабораторні роботи з дисципліни

«Програмування в середовищі Java»

**Виконав:**

Студент гр. 125-20-3

Самойленко Д.Д.

**Перевірив:**

**Доц. Мінєєв О. С.**

**Дніпро 2024**

**Лабораторна робота номер 0-1. Hello world/ CVS. GIT**

Встановити Intelliji Idea та Java jdk останньої версії. Cтворити maven проект та розробити в цьому проекті типову програму Hello world.  Програма повинна видавати на екран напис Hello world та закінчувати свою роботу.

Під'єднати до intellij Idea систему CVS. А саме GIT.  Створити аккаунт в хмарному середовищі github, під'єднати свій проект в intellij Idea до свого аккаунту github та завантажити  нульову лабораторну роботу на github аккаунт. Кожну нову лабораторну роботу робити в окремії гілці(з іменем лабораторної наприклад «LR\_3») а потім після того як її написали мержити гілку до мастера.

Код:

package edu.ntudp.fit.samoilenko.laba0\_1;  
  
public class Greeting {  
 public static void main(String[] args) {  
 System.*out*.println("Hello World!!!");  
 }  
}

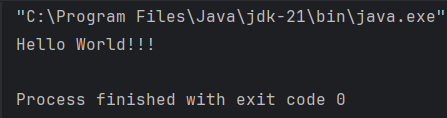


Рисунок 1 – Виконання програми Лабораторна робота 0-1

**Лабораторна робота номер 2. Основи.**

 Розробити програму, що дозволить вам створити, як з клавіатури  так і рандомно матрицю цілих чисел типу int заданої ширини та висоти(ввести з клавіатури), але не більше 20 на 20. Створити можливість пошуку в цій матриці мінімального і максимального елементу  та розрахунок середнього арифметичного. Програма може бути написана в одному класі, обов'язково розбиття на методи. Обов'язкове використання клавіатури, під час вибору ручного чи рандомного створення матриці. Створення системи зчитування з клавіатури зробити будь-яким способом, наприклад завдяки класу Scanner. Scanner являє собою найпростішу систему сканування клавіатури. Диапазон рандомних чисел для створення елементів матриці повинен зверігатись в спеціальних константах.

*Як завдання підвищеної складності додати  розрахунок середнього геометричного елементів матриці.*

Код:

package edu.ntudp.fit.samoilenko.laba2;  
import java.util.Scanner;  
  
public class MatrixOperations {  
 public static final int *MAX\_SIZE* = 20;  
 public static final int *MIN\_SIZE* = 1;  
 public static final int *MIN\_RANDOM* = 10;  
 public static final int *MAX\_RANDOM* = 25;  
  
 public static void main(String[] args) {  
 Scanner in = new Scanner(System.*in*);  
  
 System.*out*.println("Please enter size matrix");  
 System.*out*.print("Enter number of columns: ");  
 int x = *matrixSizeValidation*(in, "columns");  
  
 System.*out*.print("Enter number of rows: ");  
 int y = *matrixSizeValidation*(in, "rows");  
  
 System.*out*.println("Choose how to create the matrix:");  
 System.*out*.println("1. Manually");  
 System.*out*.println("2. Randomly");  
 System.*out*.print("Enter your choice: ");  
 int choice = in.nextInt();  
  
 switch (choice) {  
 case 1:  
 int[][] matrixManually = *createMatrixManually*(x, y, in);  
 *printMatrix*(matrixManually);  
 *calculateAndPrint*(matrixManually);  
 break;  
 case 2:  
 int[][] matrixRandomly = *createMatrixRandomly*(x, y);  
 *printMatrix*(matrixRandomly);  
 *calculateAndPrint*(matrixRandomly);  
 break;  
 default:  
 System.*out*.println("Invalid choice. Exiting...");  
 break;  
 }  
 }  
  
 public static int matrixSizeValidation(Scanner in, String element) {  
 int num = in.nextInt();  
 element = element.substring(0, 1).toUpperCase() + element.substring(1);  
  
 while (num > *MAX\_SIZE* || num < *MIN\_SIZE*) {  
 System.*out*.println(element + " must be between " + *MIN\_SIZE* + " and " + *MAX\_SIZE*);  
 System.*out*.print("Please enter other count: ");  
 num = in.nextInt();  
 }  
  
 return num;  
 }  
  
 public static void printMatrix(int[][] matrix) {  
 System.*out*.println("Matrix: ");  
 for (int i = 0; i < matrix.length; i++) {  
 for (int j = 0; j < matrix[i].length; j++) {  
 System.*out*.print(matrix[i][j] + " ");  
 }  
 System.*out*.println();  
 }  
 }  
  
 private static void calculateAndPrint(int[][] matrix) {  
 int min = *getMinValue*(matrix);  
 int max = *getMaxValue*(matrix);  
 double average = *getAverage*(matrix);  
 double geometricMean = *getGeometricMean*(matrix);  
  
 System.*out*.println("Minimum matrix element: " + min);  
 System.*out*.println("Maximum matrix element: " + max);  
 System.*out*.println("Matrix average: " + average);  
 System.*out*.println("Geometric mean: " + geometricMean);  
 }  
  
 public static int[][] createMatrixRandomly(int sizeX, int sizeY) {  
 int amountOfRandom = *MAX\_RANDOM* + *MIN\_RANDOM* + 1;  
 int[][] matrix = new int[sizeX][sizeY];  
  
 for (int i = 0; i < sizeX; i++) {  
 for (int j = 0; j < sizeY; j++) {  
 matrix[i][j] = (int) Math.*round*(Math.*random*() \* amountOfRandom - *MIN\_RANDOM*);  
 }  
 }  
  
 return matrix;  
 }  
  
 public static int[][] createMatrixManually(int sizeX, int sizeY, Scanner in) {  
 int[][] matrix = new int[sizeX][sizeY];  
  
 for (int i = 0; i < sizeX; i++) {  
 for (int j = 0; j < sizeY; j++) {  
 System.*out*.print("Enter A[" + (i + 1) + ", " + (j + 1) + "] = ");  
 matrix[i][j] = in.nextInt();  
 }  
 }  
  
 return matrix;  
 }  
  
 public static int getMinValue(int[][] matrix) {  
 int min = matrix[0][0];  
  
 for (int i = 0; i < matrix.length; i++) {  
 for (int j = 0; j < matrix[0].length; j++) {  
 if (min >= matrix[i][j]) {  
 min = matrix[i][j];  
 }  
 }  
 }  
  
 return min;  
 }  
  
 public static int getMaxValue(int[][] matrix) {  
 int max = matrix[0][0];  
  
 for (int i = 0; i < matrix.length; i++) {  
 for (int j = 0; j < matrix[0].length; j++) {  
 if (max <= matrix[i][j]) {  
 max = matrix[i][j];  
 }  
 }  
 }  
  
 return max;  
 }  
  
 public static double getAverage(int[][] matrix) {  
 int rows = matrix.length;  
 int columns = matrix[0].length;  
 int sum = 0;  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < columns; j++) {  
 sum += matrix[i][j];  
 }  
 }  
  
 return (double) sum / (rows \* columns);  
 }  
  
 public static double getGeometricMean(int[][] matrix) {  
 int rows = matrix.length;  
 int columns = matrix[0].length;  
 int multiply = 1;  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < columns; j++) {  
 multiply \*= matrix[i][j];  
 }  
 }  
  
 double geometricMean = Math.*pow*(multiply, 1.0 / (rows \* columns));  
 return geometricMean;  
 }  
}

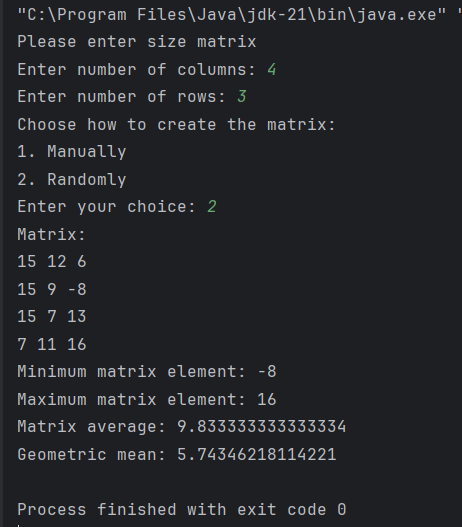


Рисунок 2 – Виконання програми Лабораторна робота 2

**Лабораторна робота номер 3. ООП.**

 Створити програму що буде створювати та обробляти комплексний об'єкт під назвою університет(university). Програма повинна складатися з трьох частин: модель вид та контролер згідно з парадигмою mvc (Model View Controller).  Кожній з цих груп повинна відповідати package з  відповідною назвою.  В моделі повинні знаходитись усі класи що відповідають за структурні підрозділи університету. Серед них:   університет, факультет, кафедра, група, студент, людина (Human). Усі вони повинні містити  назву  типу string та голову типу Human. Студент також повинен бути породжений від  Human.  Human повинен мати поля ім'я, прізвище, побатькові та стать.  Усі поля повинні бути строковими окрім поля стать. Стать повинна використовувати спеціальний  enum  типу Sex(стать).

В цій лабораторній роботі  група View Нам не потрібна.

 Що стосується групи контроллер (controller)  то вона повинна містити менеджери що дозволяють нам створити відповідні підрозділи наприклад  StudentCreator, FacultyCreator, GroupCreator  та інші, кожен з яких повинен використовувати можливості нижчого за рівнем   створювача. Програма повинна також містити клас Run, в якому буде знаходитись точка входу та методи, що повинні дати можливість створити університет.   Процес створення університету повинен бути зроблений в методі createTypycalUniversity.

  В програмі активно рекомендується використовувати абстрактні класи та інтерфейси

Код:

**Model:**

package edu.ntudp.fit.samoilenko.laba3.model;  
  
import java.util.List;  
import java.util.Objects;  
  
public class Department {  
 private Human headOfDepartment;  
 private int maxGroupsCount;  
 private List<Group> groups;  
  
 public Department(Human headOfDepartment, int maxGroupsCount, List<Group> groups) {  
 this.headOfDepartment = headOfDepartment;  
 this.maxGroupsCount = maxGroupsCount;  
 this.groups = groups;  
 }  
  
 public Human getHeadOfDepartment() {  
 return headOfDepartment;  
 }  
 public void setHeadOfDepartment(Human headOfDepartment) {  
 this.headOfDepartment = headOfDepartment;  
 }  
  
 public int getMaxGroupsCount() {  
 return maxGroupsCount;  
 }  
 public void setMaxGroupsCount(int maxGroupsCount) {  
 this.maxGroupsCount = maxGroupsCount;  
 }  
  
 public List<Group> getGroups() {  
 return groups;  
 }  
 public void setGroups(List<Group> groups) {  
 this.groups = groups;  
 }  
  
 @Override  
 public String toString() {  
 return String.*format*("\nDepartment:{headOfDepartment=%s, maxGroupsCount=%s, groups=%s}", headOfDepartment, maxGroupsCount, groups);  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (!(o instanceof Department)) return false;  
 Department department = (Department) o;  
 return maxGroupsCount == department.maxGroupsCount &&  
 headOfDepartment.equals(department.headOfDepartment) &&  
 groups.equals(department.groups);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(headOfDepartment, maxGroupsCount, groups);  
 }  
}

package edu.ntudp.fit.samoilenko.laba3.model;  
  
import java.util.List;  
import java.util.Objects;  
  
public class Faculty {  
 private Human dean;  
 private int maxDepartmentsCount;  
 private List<Department> departments;  
  
 public Faculty(Human dean, int maxDepartmentsCount, List<Department> departments) {  
 this.dean = dean;  
 this.maxDepartmentsCount = maxDepartmentsCount;  
 this.departments = departments;  
 }  
  
 public Human getDean() {  
 return dean;  
 }  
 public void setDean(Human dean) {  
 this.dean = dean;  
 }  
  
 public int getMaxDepartmentsCount() {  
 return maxDepartmentsCount;  
 }  
 public void setMaxDepartmentsCount(int maxDepartmentsCount) {  
 this.maxDepartmentsCount = maxDepartmentsCount;  
 }  
  
 public List<Department> getDepartments() {  
 return departments;  
 }  
 public void setDepartments(List<Department> departments) {  
 this.departments = departments;  
 }  
  
 @Override  
 public String toString() {  
 return String.*format*("\nFaculty:{dean=%s, maxDepartmentsCount=%s, departments=%s}", dean, maxDepartmentsCount, departments);  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (!(o instanceof Faculty)) return false;  
 Faculty faculty = (Faculty) o;  
 return maxDepartmentsCount == faculty.maxDepartmentsCount &&  
 dean.equals(faculty.dean) &&  
 departments.equals(faculty.departments);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(dean, maxDepartmentsCount, departments);  
 }  
}

package edu.ntudp.fit.samoilenko.laba3.model;  
  
import java.util.List;  
import java.util.Objects;  
  
public class Group {  
 private int maxStudentsCount;  
 private Human groupLeader;  
 private List<Student> students;  
  
 public Group(int maxStudentsCount, Human groupLeader, List<Student> students) {  
 this.maxStudentsCount = maxStudentsCount;  
 this.groupLeader = groupLeader;  
 this.students = students;  
 }  
  
 public Human getGroupLeader() {  
 return groupLeader;  
 }  
 public void setGroupLeader(Human groupLeader) {  
 this.groupLeader = groupLeader;  
 }  
  
 public int getMaxStudentsCount() {  
 return maxStudentsCount;  
 }  
 public void setMaxStudentsCount(int maxStudentsCount) {  
 this.maxStudentsCount = maxStudentsCount;  
 }  
  
 public List<Student> getStudents() {  
 return students;  
 }  
 public void setStudents(List<Student> students) {  
 this.students = students;  
 }  
  
 @Override  
 public String toString() {  
 return String.*format*("\nGroup:{groupLeader=%s, maxStudentsCount=%s, students=%s}", groupLeader, maxStudentsCount, students.toString());  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (!(o instanceof Group)) return false;  
 Group group = (Group) o;  
 return maxStudentsCount == group.maxStudentsCount &&  
 groupLeader.equals(group.groupLeader) &&  
 students.equals(group.students);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(groupLeader, maxStudentsCount, students);  
 }  
}

package edu.ntudp.fit.samoilenko.laba3.model;  
  
import java.util.Objects;  
  
public class Human {  
 protected String firstName;  
 protected String lastName;  
 protected String patronymic;  
 protected Sex sex;  
  
 public Human(String firstName, String lastName, String patronymic, Sex sex) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.patronymic = patronymic;  
 this.sex = sex;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
 public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
 public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
  
 public String getPatronymic() {  
 return patronymic;  
 }  
 public void setPatronymic(String patronymic) {  
 this.patronymic = patronymic;  
 }  
  
 public Sex getSex() {  
 return sex;  
 }  
 public void setSex(Sex sex) {  
 this.sex = sex;  
 }  
  
 @Override  
 public String toString() {  
 return String.*format*("{firstName='%s', lastName='%s', patronymic='%s', sex='%s'}", firstName, lastName, patronymic, sex);  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (!(o instanceof Human)) return false;  
 Human human = (Human) o;  
 return Objects.*equals*(firstName, human.firstName) &&  
 Objects.*equals*(lastName, human.lastName) &&  
 Objects.*equals*(patronymic, human.patronymic) &&  
 sex == human.sex;  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(firstName, lastName, patronymic, sex);  
 }  
}

package edu.ntudp.fit.samoilenko.laba3.model;  
  
public enum Sex {  
 *MALE*,  
 *FEMALE*,  
}

package edu.ntudp.fit.samoilenko.laba3.model;  
  
import java.util.Objects;  
  
public class Student extends Human {  
 private int recordBookNumber;  
  
 public Student(String firstName, String lastName, String patronymic, Sex sex, int recordBookNumber) {  
 super(firstName, lastName, patronymic, sex);  
 this.recordBookNumber = recordBookNumber;  
 }  
  
 public int getRecordBookNumber() {  
 return recordBookNumber;  
 }  
 public void setRecordBookNumber(int recordBookNumber) {  
 this.recordBookNumber = recordBookNumber;  
 }  
  
 @Override  
 public String toString() {  
 return String.*format*("\n Student:{recordBookNumber='%s', firstName='%s', lastName='%s', patronymic='%s', sex='%s'}", recordBookNumber, firstName, lastName, patronymic, sex);  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (!(o instanceof Student)) return false;  
 Student student = (Student) o;  
 return recordBookNumber == student.recordBookNumber;  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(recordBookNumber);  
 }  
}

package edu.ntudp.fit.samoilenko.laba3.model;  
  
import java.util.List;  
import java.util.Objects;  
  
public class University {  
 private String name;  
 private Human rector;  
 private List<Faculty> faculties;  
  
 public University(String name, Human rector, List<Faculty> faculties) {  
 this.name = name;  
 this.rector = rector;  
 this.faculties = faculties;  
 }  
  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public Human getRector() {  
 return rector;  
 }  
 public void setRector(Human rector) {  
 this.rector = rector;  
 }  
  
 public List<Faculty> getFaculties() {  
 return faculties;  
 }  
 public void setFaculties(List<Faculty> faculties) {  
 this.faculties = faculties;  
 }  
  
 @Override  
 public String toString() {  
 return String.*format*("University{name='%s', rector=%s, faculties=%s}", name, rector, faculties);  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (!(o instanceof University)) return false;  
 University university = (University) o;  
 return Objects.*equals*(name, university.name) && Objects.*equals*(rector, university.rector) && Objects.*equals*(faculties, university.faculties);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(name, rector, faculties);  
 }  
}

**Controller:**

package edu.ntudp.fit.samoilenko.laba3.controller;  
  
import edu.ntudp.fit.samoilenko.laba3.model.Department;  
import edu.ntudp.fit.samoilenko.laba3.model.Group;  
import edu.ntudp.fit.samoilenko.laba3.model.Human;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class DepartmentCreator {  
  
 public Department createDepartment(int countOfGroups, int maxGroupsCount, Human headOfDepartment) {  
 List<Group> groups = new ArrayList<>();  
 GroupCreator groupCreator = new GroupCreator();  
 Department department = new Department(headOfDepartment, maxGroupsCount, groups);  
  
 for (int i = 0; i < countOfGroups; i++) {  
 groups.add(groupCreator.createTypicalGroup());  
 }  
  
 return department;  
 }  
  
 public Department createTypicalDepartment() {  
 HumanCreator headOfDepartment = new HumanCreator();  
 return createDepartment(5, 6, headOfDepartment.createTypicalHuman());  
 }  
}

package edu.ntudp.fit.samoilenko.laba3.controller;  
  
import edu.ntudp.fit.samoilenko.laba3.model.Department;  
import edu.ntudp.fit.samoilenko.laba3.model.Faculty;  
import edu.ntudp.fit.samoilenko.laba3.model.Human;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class FacultyCreator {  
  
 public Faculty createFaculty(int countOfDepartments, int maxDepartmentsCount, Human dean) {  
 List<Department> departments = new ArrayList<>();  
 DepartmentCreator departmentCreator = new DepartmentCreator();  
 Faculty faculty = new Faculty(dean, maxDepartmentsCount, departments);  
  
 for (int i = 0; i < countOfDepartments; i++) {  
 departments.add(departmentCreator.createTypicalDepartment());  
 }  
  
 return faculty;  
 }  
  
 public Faculty createTypicalFaculty() {  
 HumanCreator dean = new HumanCreator();  
 Faculty group = createFaculty(3, 4, dean.createTypicalHuman());  
 return group;  
 }  
}

package edu.ntudp.fit.samoilenko.laba3.controller;  
  
import edu.ntudp.fit.samoilenko.laba3.model.Group;  
import edu.ntudp.fit.samoilenko.laba3.model.Human;  
import edu.ntudp.fit.samoilenko.laba3.model.Student;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class GroupCreator {  
 public Group createGroup(int countOfStudents, int maxStudentsCount, Human groupLeader) {  
 List<Student> students = new ArrayList<>();  
 StudentCreator studentCreator = new StudentCreator();  
 Group group = new Group(maxStudentsCount, groupLeader, students);  
  
 for (int i = 0; i < countOfStudents; i++) {  
 students.add(studentCreator.createTypicalStudent());  
 }  
  
 return group;  
 }  
  
 public Group createTypicalGroup() {  
 HumanCreator groupLeader = new HumanCreator();  
 return createGroup(5, 20, groupLeader.createTypicalHuman());  
 }  
  
}

package edu.ntudp.fit.samoilenko.laba3.controller;  
  
import edu.ntudp.fit.samoilenko.laba3.model.Human;  
import edu.ntudp.fit.samoilenko.laba3.model.Sex;  
  
public class HumanCreator {  
 public Human createHuman(String firstName, String lastName, String patronymic, Sex sex) {  
  
 return new Human(firstName, lastName, patronymic, sex);  
 }  
  
 public Human createTypicalHuman(){  
 return createHuman("Diana", "Samoilenko", "Dmitrievna", Sex.*FEMALE*);  
 }  
}

package edu.ntudp.fit.samoilenko.laba3.controller;  
  
import edu.ntudp.fit.samoilenko.laba3.model.Human;  
import edu.ntudp.fit.samoilenko.laba3.model.Student;  
  
public class StudentCreator {  
 public Student createStudent(Human human, int recordBookNumber) {  
  
 return new Student(human.getFirstName(), human.getLastName(), human.getPatronymic(), human.getSex(), recordBookNumber);  
 }  
  
 public Student createTypicalStudent() {  
 HumanCreator human = new HumanCreator();  
 Student typicalStudent = createStudent(human.createTypicalHuman(), 33337777);  
 return typicalStudent;  
 }  
  
}

package edu.ntudp.fit.samoilenko.laba3.controller;  
  
import edu.ntudp.fit.samoilenko.laba3.model.Faculty;  
import edu.ntudp.fit.samoilenko.laba3.model.Human;  
import edu.ntudp.fit.samoilenko.laba3.model.University;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class UniversityCreator {  
 public University createUniversity(String name, int countOfFaculties, Human rector) {  
 List<Faculty> faculties = new ArrayList<>();  
 University university = new University(name, rector, faculties);  
 FacultyCreator facultyCreator = new FacultyCreator();  
  
 for (int i = 1; i < countOfFaculties; i++) {  
 faculties.add(facultyCreator.createTypicalFaculty());  
 }  
  
 return university;  
 }  
  
 public University createTypicalUniversity() {  
 HumanCreator rector = new HumanCreator();  
 University university = createUniversity("NTU DP", 4, rector.createTypicalHuman());  
  
 return university;  
 }  
  
}

Run:

package edu.ntudp.fit.samoilenko.laba3;  
  
import edu.ntudp.fit.samoilenko.laba3.controller.UniversityCreator;  
import edu.ntudp.fit.samoilenko.laba3.model.University;  
  
public class Run {  
 public static void main(String[] args) {  
 UniversityCreator universityCreator = new UniversityCreator();  
 University university = universityCreator.createTypicalUniversity();  
  
 System.*out*.println(university.toString());  
 }  
}

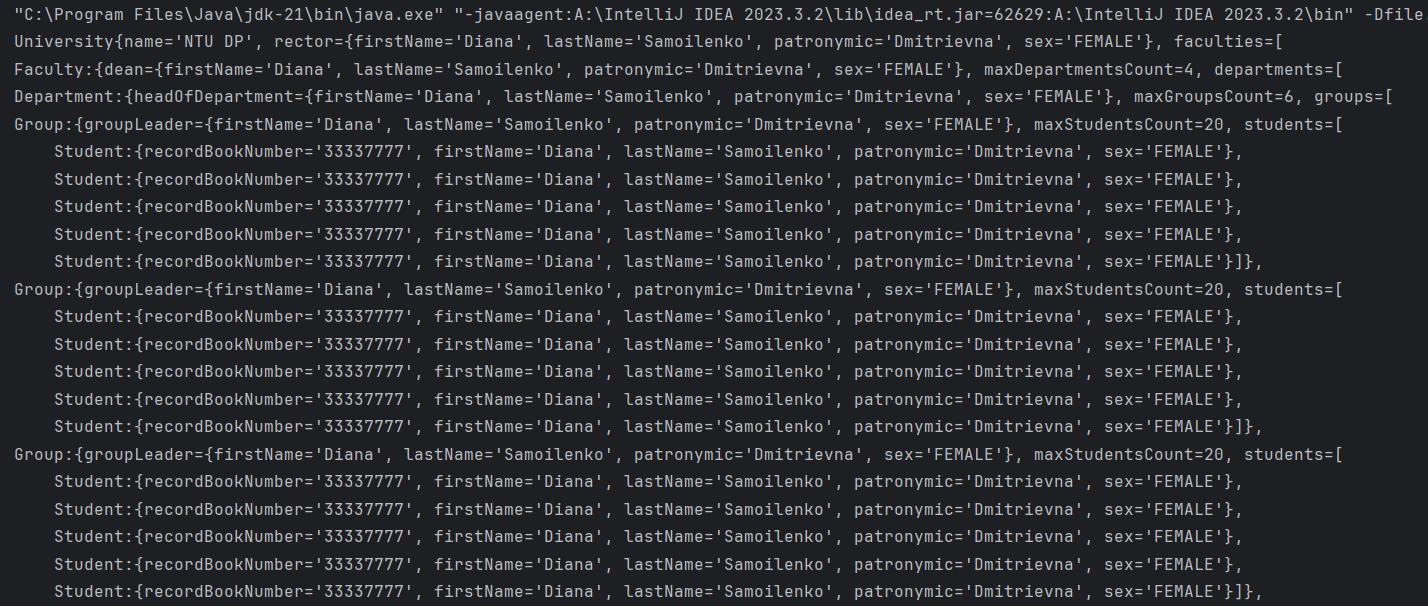


Рисунок 3 – Виконання програми Лабораторна робота 3

**Лабораторна робота номер 4. JUnit. Json**

 Додати до лабораторної роботи 3 можливість запису університету  у формат json, запис цього формату у файл, зчитування цього формату файлу, та створення об'єкту з текстового формату json.  В  проекті повинен бути зроблений JUnit тест,  який буде виглядати наступним чином:  створити об'єкт університет(oldUniversity),  в якому в кожному підрозділі маються два підрозділи нижчого рівня. Наприклад на факультеті дві кафедри, на кожній кафедрі дві групи, на кожній групі два студенти.  Цей об'єкт повинен бути записаний в файл у форматі json. Потім з цього файлу зчитаний та відновлений як newUniversity. В тесті повинні бути порівняні newUniversity та oldUniversity за допомогою методу equals.  Якщо все зроблено правильно то університети повинні бути еквівалентні, а метод equals повинен повернути True.  Для запису та зчитування університету у форматі json повинен бути зроблений клас JsonManager.  Для безпосереднього перетворення університету  у формат json та його відновлення цього формату, можливо використання сторонніх бібліотек наприклад Gson, Jackson  чи будь-яких інших.

Для початку розробки лабораторної роботи номер 4 повністю скопіювати програмний код лабораторної роботи номер 3. Не змішувати ці роботи не в якому разі.

Код:

package edu.ntudp.fit.samoilenko.laba4;  
  
import com.google.gson.Gson;  
import edu.ntudp.fit.samoilenko.laba4.model.University;  
  
import java.io.FileNotFoundException;  
import java.io.FileReader;  
import java.io.FileWriter;  
import java.io.IOException;  
  
public class JsonManager {  
  
 public void writeToJson(Object o, String filePath) {  
 Gson gson = new Gson();  
 String json = gson.toJson(o);  
  
 try {  
 FileWriter writer = new FileWriter(filePath);  
 writer.write(json);  
 writer.close();  
  
 System.*out*.println("Successful write to file");  
 } catch (IOException e) {  
 System.*out*.println("There was an error");  
 e.printStackTrace();  
 }  
 }  
  
 public Object readFromJson(String filePath) {  
 Gson gson = new Gson();  
  
 try {  
 return gson.fromJson(new FileReader(filePath), University.class);  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Error");  
 e.printStackTrace();  
 }  
  
 return null;  
 }  
}

package edu.ntudp.fit.samoilenko.laba4;  
  
import edu.ntudp.fit.samoilenko.laba4.controller.UniversityCreator;  
import edu.ntudp.fit.samoilenko.laba4.model.University;  
  
public class Run {  
 public static void main(String[] args) {  
 UniversityCreator universityCreator = new UniversityCreator();  
 University university = universityCreator.createTypicalUniversity();  
  
 System.*out*.println(university.toString());  
 }  
}



Рисунок 4 – Вміст файлу university.json з Лабораторної роботи 4

**Лабораторна робота номер 5. Jdbc**

 Створити базу даних в будь  якому сервері баз даних. Створити таблицю з переліком студентів вказати їх прізвище, ім'я, по батькові, день народження номер залікової книжки та ID.

Створити програму  що буде дозволяти   виводити на екран  інформацію про студентів  які народилися в тому чи іншому місяці року.  Програма повинна завдяки системі jdbc під'єднатися до вашої бази даних та робити до неї запроси.  Вимог до розробки бази даних немає.  Програма ж має бути написана за усіма стандартами ООП.  Та може бути спроектована за двох принципів:

- при будь-якій ситуації буде забиратися весь перелік студентів, а вже на стороні java буде зроблено пошук  необхідного

 - SQL запрос буде сформований згідно запросу який зробив користувач і вже сервер управління баз даних буде вирішувати, які самі студенти народилися в тому чи іншому місяці.

 У висновку обов'язково пояснити чому вибрали той чи інший принцип, які в нього переваги та недоліки.  Оцінка не залежить від того який сервер управління баз даних вибрали.  Перелік студентів зробити не менше 20 людей. Місяць червень зробити місяцем, коли в жодного зі студентів немає дня народження.

 SQL код створення бази даних розмістити проекті 6 лабораторної роботи  в файлі database в пекеджі resources.  Для використання цієї лабораторної роботи рекомендується активно використовувати знання отримані на дисципліні що стосуються  розробки баз даних.

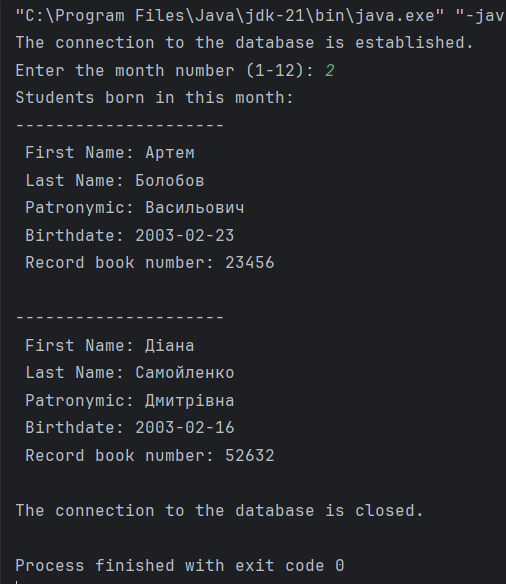
 До паперового звіту обов'язково додати принтскрин з програми  в якій ви дивитесь інформацію  вашого  сервера управління баз даних, де показати створену таблицю, її ім'я та загальні відомості бази даних, наприклад назва, ім'я, назва користувача адміністратора, пароль тощо.  Для роботи з сервером управління баз даних рекомендуємо використовувати програмне забезпечення компанії jetbrains datagrip. Або вбудовану панель користування базами даних, що міститься у середовищі intellij Idea,  яка на сьогоднішній день підтримує майже всі сервери управління баз даних.

Код:

package edu.ntudp.fit.samoilenko.laba5;  
  
import java.sql.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
public class DBManager {  
 private static final String *URL* = "jdbc:mysql://localhost:3306/StudentDB";  
 private static final String *USER* = "diana";  
 private static final String *PASSWORD* = "81v8p9atpsvn";  
 private static Connection *connection*;  
  
 public void connect() throws SQLException {  
 *connection* = DriverManager.*getConnection*(*URL*, *USER*, *PASSWORD*);  
 if(!*connection*.isClosed()) {  
 System.*out*.println("The connection to the database is established.");  
 }  
 }  
  
 public List<Student> getAllStudents() throws SQLException {  
 String sqlQuery = "SELECT \* FROM students";  
  
 try (Statement statement = *connection*.createStatement();  
 ResultSet resultSet = statement.executeQuery(sqlQuery)) {  
 return getStudents(resultSet);  
 }  
 }  
  
 public List<Student> getStudentsByBirthMonth(int month) throws SQLException {  
 String sqlQuery = "SELECT \* FROM students WHERE MONTH(birthdate) = ?";  
  
 try (PreparedStatement statement = *connection*.prepareStatement(sqlQuery)) {  
 statement.setInt(1, month);  
 try (ResultSet resultSet = statement.executeQuery()) {  
 return getStudents(resultSet);  
 }  
 }  
 }  
  
 private Student createStudent(ResultSet resultSet) throws SQLException {  
 return new Student(  
 resultSet.getInt("id"),  
 resultSet.getString("firstName"),  
 resultSet.getString("lastName"),  
 resultSet.getString("patronymic"),  
 resultSet.getDate("birthDate"),  
 resultSet.getInt("recordBookNumber")  
 );  
 }  
  
 private List<Student> getStudents(ResultSet resultSet) throws SQLException {  
 List<Student> students = new ArrayList<>();  
 while (resultSet.next()) {  
 students.add(createStudent(resultSet));  
 }  
 return students;  
 }  
  
 public void closeConnection() throws SQLException {  
 if (*connection* != null && !*connection*.isClosed()) {  
 *connection*.close();  
 System.*out*.println("The connection to the database is closed.");  
 }  
 }  
}

package edu.ntudp.fit.samoilenko.laba5;  
  
import java.util.Date;  
  
public class Student {  
 private int id;  
 private String firstName;  
 private String lastName;  
 private String patronymic;  
 private Date birthDate;  
 private int recordBookNumber;  
  
 public Student(int id, String firstName, String lastName, String patronymic, Date birthDate, int recordBookNumber) {  
 this.id = id;  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.patronymic = patronymic;  
 this.birthDate = birthDate;  
 this.recordBookNumber = recordBookNumber;  
 }  
  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
  
 public String getPatronymic() {  
 return patronymic;  
 }  
  
 public void setPatronymic(String middleName) {  
 this.patronymic = patronymic;  
 }  
  
 public Date getBirthDate() {  
 return birthDate;  
 }  
  
 public void setBirthDate(Date birthDate) {  
 this.birthDate = birthDate;  
 }  
  
 public int getRecordBookNumber() {  
 return recordBookNumber;  
 }  
  
 public void setRecordBookNumber(int recordBookNumber) {  
 this.recordBookNumber = recordBookNumber;  
 }  
  
 @Override  
 public String toString() {  
 return String.*format*(" First Name: %s\n Last Name: %s\n Patronymic: %s\n Birthdate: %s\n Record book number: %s\n", firstName, lastName, patronymic, birthDate, recordBookNumber);  
 }  
}

import java.sql.SQLException;  
import java.util.List;  
import java.util.Scanner;  
  
public class Run {  
 public static void main(String[] args) throws SQLException {  
 Scanner scanner = new Scanner(System.*in*);  
  
 DBManager dbManager = new DBManager();  
 dbManager.connect();  
  
 int month = *validationOfBirthMonth*(scanner);  
  
 List<Student> filteredStudents = dbManager.getStudentsByBirthMonth(month);  
 *printStudentsByMonth*(filteredStudents);  
  
 dbManager.closeConnection();  
 }  
  
 private static int validationOfBirthMonth(Scanner scanner) {  
 int month;  
  
 do {  
 System.*out*.print("Enter the month number (1-12): ");  
 month = Integer.*parseInt*(scanner.nextLine());  
  
 if (month < 1 || month > 12) {  
 System.*out*.println("Invalid month number. Please enter a number between 1 and 12.");  
 }  
 } while (month < 1 || month > 12);  
  
 return month;  
 }  
  
 private static void printStudentsByMonth(List<Student> students) {  
 String message = students.isEmpty() ?  
 "No students born this month were found..." :  
 "Students born in this month:";  
 System.*out*.println(message);  
  
 if (!students.isEmpty()) {  
 for (Student student : students) {  
 System.*out*.println("---------------------");  
 System.*out*.println(student.toString());  
 }  
 }  
 }  
}



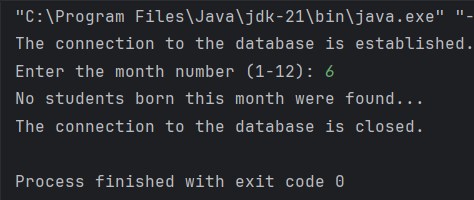


Рисунок 5 – Виконання програми Лабораторна робота 5

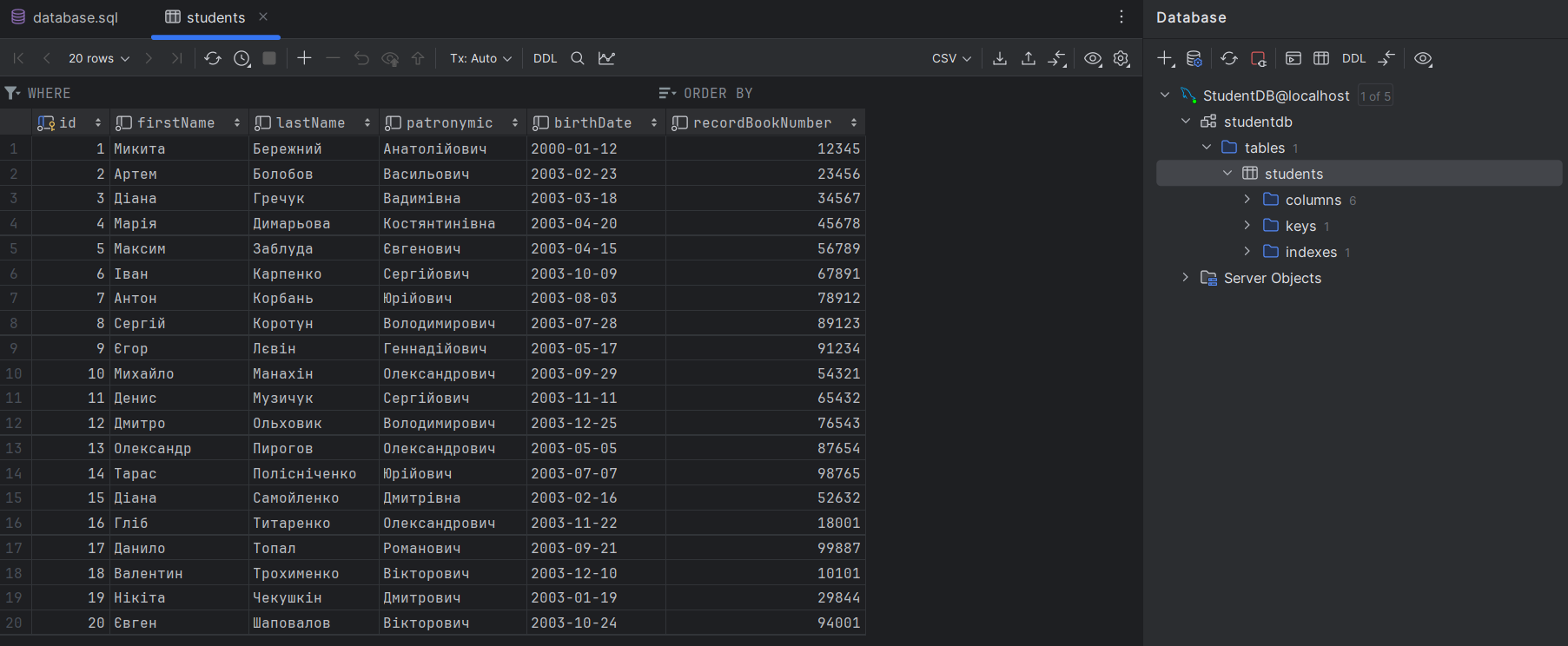


Рисунок 6 – Таблиця бази даних

Було реалізовано два методи класу DBManager для вибірки студентів за бази даних:

1. getAllStudents() – Для отримання всіх студентів з бази даних – «SELECT \* FROM students».
2. getStudentsByBirthMonth() – Для фільтрації студентів за місяцем народження я обрала окремий запит sql – «SELECT \* FROM students WHERE MONTH(birthdate) = ?».

Цей метод більш ефективний так як SQL-запит дозволяє базі даних обробляти і фільтрувати дані безпосередньо на рівні сервера баз даних. SQL-запит зменшує навантаження на мережу, адже передається лише необхідний обсяг даних, а саме студенти, які народжені в певний місяць року. Також запит SQL досить простий і короткий для вибірки необхідних даних. Це дозволяє зберегти код програми більш чистим і зрозумілим, оскільки весь процес фільтрації відбувається на рівні бази даних.

Посилання на гіт-хаб репозиторій: <https://github.com/dianasamoilenko/java-labs.git>