НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №1

з дисципліни

“Програмування мобільних систем / Розроблення клієнтських додатків для мобільних платформ”

Виконав:

студент групи ІО-83

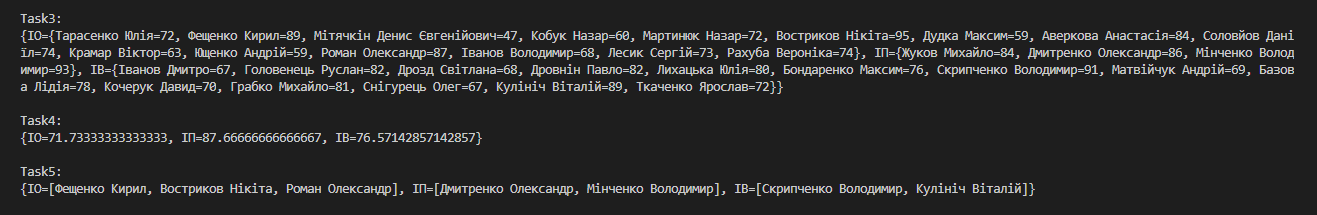
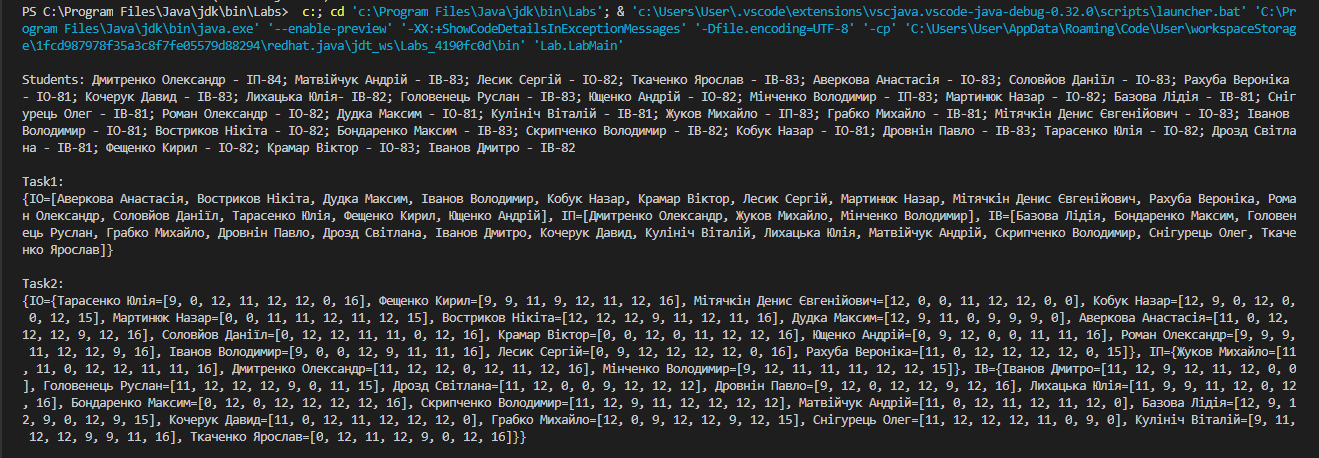
ЗК ІО-8313

Коломієць Дмитро

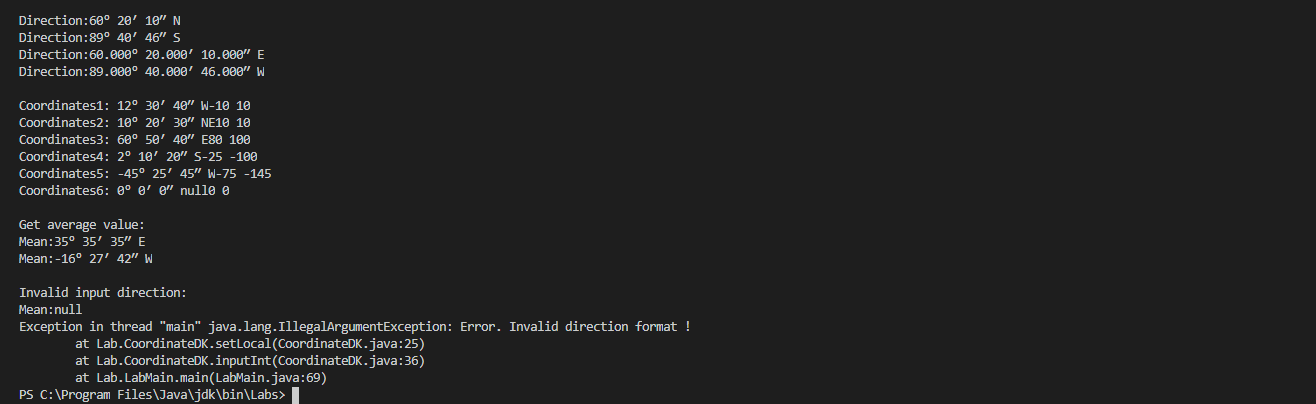
Київ 2021

**Скріншот роботи програми:**

1 завдання:



2 завдання:



**Лістинг коду:**

**LabMain.java**

package Lab;

import java.util.List;

import java.util.Map;

public class LabMain {

        public static void main(String[] args) {

            String studentStr = "Дмитренко Олександр - ІП-84; Матвійчук Андрій - ІВ-83; Лесик Сергій - ІО-82; Ткаченко Ярослав - ІВ-83; Аверкова Анастасія - ІО-83; Соловйов Даніїл - ІО-83; Рахуба Вероніка - ІО-81; Кочерук Давид - ІВ-83; Лихацька Юлія- ІВ-82; Головенець Руслан - ІВ-83; Ющенко Андрій - ІО-82; Мінченко Володимир - ІП-83; Мартинюк Назар - ІО-82; Базова Лідія - ІВ-81; Снігурець Олег - ІВ-81; Роман Олександр - ІО-82; Дудка Максим - ІО-81; Кулініч Віталій - ІВ-81; Жуков Михайло - ІП-83; Грабко Михайло - ІВ-81; Мітячкін Денис Євгенійович - ІО-83; Іванов Володимир - ІО-81; Востриков Нікіта - ІО-82; Бондаренко Максим - ІВ-83; Скрипченко Володимир - ІВ-82; Кобук Назар - ІО-81; Дровнін Павло - ІВ-83; Тарасенко Юлія - ІО-82; Дрозд Світлана - ІВ-81; Фещенко Кирил - ІО-82; Крамар Віктор - ІО-83; Іванов Дмитро - ІВ-82";

            Integer[] points = {12, 12, 12, 12, 12, 12, 12, 16};

            StudentsParse studentsParse = new StudentsParse(studentStr);

            System.out.println("\nStudents: " + studentStr);

            //Task 1

            Map<String, List<Student>> groupMaps = studentsParse.takeOfGroups();

            System.out.println("\nTask1:\n" + groupMaps);

            //Task2

            Map<String, Map<Student, List<Integer>>> theGroupWithMarks = studentsParse.theGroupWithMarks(points);

            System.out.println("\nTask2:\n" + theGroupWithMarks);

            //Task3

            Map<String, Map<Student, Integer>> groupOfMiddleFinalMarks = studentsParse.groupOfMiddleFinalMarks(theGroupWithMarks);

            System.out.println("\nTask3:\n" + groupOfMiddleFinalMarks);

            //Task4

            Map<String, Double> groupOfMiddleMarks = studentsParse.groupOfMiddleMarks(groupOfMiddleFinalMarks);

            System.out.println("\nTask4:\n" + groupOfMiddleMarks);

            //Task5

            int max = 85;

            Map<String, List<Student>> filtStudents = studentsParse.filtStudents(groupOfMiddleFinalMarks, max);

            System.out.println("\nTask5:\n" + filtStudents);

            CoordinateDK coordinateDK = new CoordinateDK(10, 20, 30, "E");

            Direction direction = new Direction(10, 10);

            CoordinateDK[] cor2 = {

                new CoordinateDK(12, 30, 40, "W"),

                new CoordinateDK(10, 20, 30, "NE"),

                new CoordinateDK(60, 50, 40, "E"),

                new CoordinateDK(2, 10, 20, "S"),

                new CoordinateDK(-45, 25, 45, "W"),

                new CoordinateDK()

            };

            Direction[] dir2 = {

                new Direction(-10, 10),

                new Direction(10, 10),

                new Direction(80, 100),

                new Direction(-25, -100),

                new Direction(-75, -145),

                new Direction()

            };

            System.out.println("\nDirection:"+ coordinateDK.inputInt(60, 20, 10) + " " + direction.inputDirection(0, 110));

            System.out.println("Direction:"+ coordinateDK.inputInt(89, 40, 46) + " " + direction.inputDirection(0, -50));

            System.out.println("Direction:"+ coordinateDK.inputFloat(60.0, 20.0, 10.0) + " " + direction.inputDirection(80, 0));

            System.out.println("Direction:"+ coordinateDK.inputFloat(89.0, 40.0, 46.0) + " " + direction.inputDirection(-10, 0) +"\n");

            for (int i = 0; i < cor2.length; i++)

                System.out.printf("Coordinates%d: %s%s%n", i+1, cor2[i], dir2[i]) ;

            System.out.println("\nGet average value:");

            System.out.println("Mean:"+ coordinateDK.getOne(cor2[2]));

            System.out.println("Mean:"+ CoordinateDK.getmiddle(cor2[0], cor2[4]));

            System.out.println("\nInvalid input direction:");

            System.out.println("Mean:"+ coordinateDK.getOne(cor2[1]));

            System.out.println("Direction:"+ coordinateDK.inputInt(60, -1, 10) + " " + direction.inputDirection(0, 110));

        }

}

**CoordinateDK.java**

package Lab;

public class CoordinateDK {

    private String direction;

    private int degrees;

    private int minutes;

    private int seconds;

    int subv;

    public CoordinateDK(){

        this.seconds = 0;

        this.minutes = 0;

        this.degrees = 0;

    }

    public CoordinateDK(int degrees, int minutes, int seconds, String direction){

        this.seconds = seconds;

        this.minutes = minutes;

        this.degrees = degrees;

        this.direction = direction;

    }

    private void setLocal(int degrees,  int minutes, int seconds){

        if ((degrees > 90 || degrees < -90) || (minutes < 0 || minutes > 59) || (seconds < 0 || seconds > 59)){

            throw new IllegalArgumentException("Error. Invalid direction format !");

        }

    }

    private void setLocalf(double degrees,  double minutes, double seconds){

        if ((degrees > 90.0 || degrees < -90.0) || (minutes < 0 || minutes > 59.9) || (seconds < 0 || seconds > 59.9)){

            throw new IllegalArgumentException("Error. Invalid direction format !");

        }

    }

    public String inputInt(int degrees, int minutes, int seconds){

        setLocal(degrees, minutes, seconds);

        String taskA = String.format("%d° %d′ %d″", degrees, minutes, seconds);

        return taskA;

    }

    public String inputFloat(double degrees, double minutes, double seconds){

        setLocalf(degrees, minutes, seconds);

        String taskB = String.format("%.3f° %.3f′ %.3f″", degrees, minutes, seconds);

        return taskB;

    }

    CoordinateDK getOne(CoordinateDK cor2){

        if ((direction.length() > 1) || (cor2.direction.length() > 1))

            return null;

        else if (direction.length() <= 1){

            if ((direction == "N") && (cor2.direction == "S") || (direction == "S") && (cor2.direction == "N")){

                degrees = this.degrees - cor2.degrees;

                minutes = this.minutes - cor2.minutes;

                seconds = this.seconds - cor2.seconds;}

            else if ((direction == "W") && (cor2.direction == "E") || (direction == "E") && (cor2.direction == "W")){

                degrees = this.degrees - cor2.degrees;

                minutes = this.minutes - cor2.minutes;

                seconds = this.seconds - cor2.seconds;}

            else if ((direction == "W") && (cor2.direction == "N") || (direction == "N") && (cor2.direction == "W"))

                return null;

            else if ((direction == "N") && (cor2.direction == "E") || (direction == "E") && (cor2.direction == "N"))

                return null;

            else if ((direction == "E") && (cor2.direction == "S") || (direction == "S") && (cor2.direction == "E"))

                return null;

            else if ((direction == "W") && (cor2.direction == "S") || (direction == "S") && (cor2.direction == "W"))

                return null;

            else

                degrees = (this.degrees + cor2.degrees) / 2;

                minutes = (this.minutes + cor2.minutes) / 2;

                seconds = (this.seconds + cor2.seconds) / 2;

            }

        return new CoordinateDK(degrees, minutes, seconds, direction);

    }

    public static CoordinateDK getmiddle(CoordinateDK cor1, CoordinateDK cor2){

        return cor1.getOne(cor2);

    }

    @Override

    public String toString(){

        return degrees + "° " + minutes + "′ " + seconds + "″ " + direction;

    }

}

**StudentParse.java**

package Lab;

import java.text.Collator;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collection;

import java.util.HashMap;

import java.util.HashSet;

import java.util.List;

import java.util.Locale;

import java.util.Map;

import java.util.Set;

import java.util.stream.Collectors;

public class StudentsParse {

    private static final Collator collator = Collator.getInstance(new Locale("uk", "UA"));

    private final String studentsStr;

    private final Map<Student, String> groupOfStudens;

    private final List<Student> students;

    public StudentsParse(String studentsStr) {

        this.studentsStr = studentsStr;

        this.groupOfStudens = parse(studentsStr);

        this.students = new ArrayList<>(groupOfStudens.keySet());

    }

    public List<Student> getStudents(){

        return students;

    }

    public Collection<String> getGroups(){

        return this.groupOfStudens.values();

    }

    public String getInputStr(){

        return studentsStr;

    }

    public Map<Student, String > getGroupOfStudens(){

        return getGroupOfStudens();

    }

    private Map<Student, String> parse(String studentsStr){

        if (studentsStr == null){

            return null;

        }

        String[] s = studentsStr.split(";");

        return Arrays.stream(s).map(students -> students.split("-"))

                .collect(Collectors.toMap(strings -> new Student(strings[0].trim()), strings -> strings[1].trim()));

    }

    public Map<String, List<Student>> takeOfGroups(){

        Set<String> groups = new HashSet<>(this.getGroups());

        Map<String, List<Student>> groupMap = new HashMap<>();

        for (final String group : groups){

            List<Student> s = groupOfStudens.entrySet().stream().filter(students -> students.getValue().equals(group)).map(Map.Entry::getKey)

                    .sorted((s1, s2) -> collator.compare(s1.getName(), s2.getName())).collect(Collectors.toList());

            groupMap.put(group, s);

        }

        return groupMap;

    }

    public Map<String, Double> groupOfMiddleMarks(Map<String, Map<Student, Integer>> studentsWithFinalMarks) {

        Map<String, Double> groupAvgMarks = new HashMap<>();

        for (Map.Entry<String, Map<Student, Integer>> studentEntrySetWithMarks: studentsWithFinalMarks.entrySet()) {

            Map<Student, Integer> studentSet = studentEntrySetWithMarks.getValue();

            Double avg = null;

            avg = studentSet.values().stream().mapToDouble(Integer::doubleValue).average().getAsDouble();

            groupAvgMarks.put(studentEntrySetWithMarks.getKey(), avg);

        }

        return groupAvgMarks;

    }

    public Map<String, Map<Student, Integer>> groupOfMiddleFinalMarks(Map<String, Map<Student, List<Integer>>> studentsWithPointsByGroups) {

        Map<String, Map<Student, Integer>> studentsWithMarksByGroup = new HashMap<>();

        for (Map.Entry<String, Map<Student, List<Integer>>> studentsWithPoints: studentsWithPointsByGroups.entrySet()) {

            Map<Student, Integer> studentsWithSumMarks = new HashMap<>();

            for (Map.Entry<Student, List<Integer>> studentIntegerMap: studentsWithPoints.getValue().entrySet()) {

                studentsWithSumMarks.put(

                        studentIntegerMap.getKey(),

                        studentIntegerMap.getValue().stream()

                                .mapToInt(Integer::intValue)

                                .sum()

                );

            }

            studentsWithMarksByGroup.put(studentsWithPoints.getKey(), studentsWithSumMarks);

        }

        return studentsWithMarksByGroup;

    }

    public Map<String, List<Student>> filtStudents(Map<String, Map<Student, Integer>> studentsWithFinalMarks, int floor) {

        Map<String, List<Student>> filteredStudentMap = new HashMap<>();

        for (Map.Entry<String, Map<Student, Integer>> studentSetWithMarks: studentsWithFinalMarks.entrySet()) {

            List<Student> students = studentSetWithMarks.getValue().entrySet().stream()

                    .filter(x -> x.getValue() >= floor)

                    .map(Map.Entry::getKey)

                    .collect(Collectors.toList());

            filteredStudentMap.put(studentSetWithMarks.getKey(), students);

        }

        return filteredStudentMap;

    }

    public Map<String, Map<Student, List<Integer>>> theGroupWithMarks(Integer[] maxPoints) {

        Map<String, Map<Student, List<Integer>>> studentsWithMarksByGroups = new HashMap<>();

        Map<String, List<Student>> groupMap = this.takeOfGroups();

        for (Map.Entry<String, List<Student>> gm : groupMap.entrySet()) {

            Map<Student, List<Integer>> studentsWithMarks = gm.getValue().stream()

                    .collect(Collectors.toMap(

                            student -> student, student -> Students.getPoints(maxPoints)

                            )

                    );

            studentsWithMarksByGroups.put(gm.getKey(), studentsWithMarks);

        }

        return studentsWithMarksByGroups;

    }

}

**Висновок:** при виконанні першого завдання було використано колекції в мові java. Також, при виконанні другого використовувались простіші методи програмування мовою java. Проблем при виконанні лабораторної роботи не виникало.