Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи №5 з дисципліни
« Основи програмування 2. Модульне програмування»

«Успадкування та поліморфізм»
Варіант 3

Виконав студент ІП-15, Борисик Владислав Тарасович
(шифр, прізвище, ім'я, по батькові)

Перевірила Вєчерковська Анастасія Сергіївна
(прізвище, ім'я, по батькові)

Київ 2022

Лабораторна робота №4

Успадкування та поліморфізм

Варіант 3

Задача

3. Створити клас TLine, що представляє пряму і містить методи для визначення того, чи є інша пряма паралельною / перпендикулярною до неї, та, чи належить вказана точка прямій. На основі цього класу створити класи-нащадки, що представляють пряму на площині і в просторі. Випадковим чином згенерувати дані для створення n прямих у просторі та m прямих на площині. Визначити, чи належить вказана точка хоча б одній прямій на площині, серед тих, які є перпендикулярними до першої (в порядку створення) прямої на площині, та, чи є серед заданих прямих у просторі така, що є перпендикулярною до всіх інших прямих у просторі.

Код

C++

main.cpp

```
#include "vector"
#include <iostream>
#include "functions.h"
#include "ctime"


int main() {
    srand(time(nullptr));

    int n = capture_n();
    int m = capture_m();

    vector<LineOnPlane> lines_2d = generate_2d_lines(n);
    vector<LineInSpace> lines_3d = generate_3d_lines(m);

    Point2D point = capture_point();

    vector<LineOnPlane> perpendicular_line =
perpendicular_lines(lines_2d);
    check_if_2dlines_contains_point(perpendicular_line, point);

    check_if_3Dlines_contains_others(lines_3d);
}
```

classes.h

```cpp
#ifndef INC_2LABWORK_5_CLASSES_H
#define INC_2LABWORK_5_CLASSES_H

#endif

using namespace std;
#include <iostream>

class TLine{
protected:
    float begin_x;
    float begin_y;
    float end_x;
    float end_y;

public:
    virtual float get_begin_x() const = 0;
    virtual float get_begin_y() const = 0;
    virtual float get_end_x() const = 0;
    virtual float get_end_y() const = 0;
};

class Point2D{
private:
    float x;
    float y;
public:
    Point2D(float x, float y);

    float get_x() const;
    float get_y() const;
};

class LineOnPlane : public TLine{
public:
    LineOnPlane(float begin_x, float begin_y, float end_x, float end_y);

    float get_begin_x() const override;
    float get_begin_y() const override;
    float get_end_x() const override;
    float get_end_y() const override;

    bool is_parallel(const LineOnPlane& line) const;
    bool is_perpendicular(const LineOnPlane& line) const;
    bool is_belongs(Point2D point) const;

    string get_info();
};

class Vector3D{
private:
    float x;
    float y;
    float z;
public:
    Vector3D(float x, float y, float z);

    float get_x() const;
    float get_y() const;
    float get_z() const;

    friend float operator * (Vector3D vec1, Vector3D vec2);
};

class LineInSpace : public TLine{
private:
    float begin_z;
    float end_z;

public:
    LineInSpace(float begin_x, float begin_y, float begin_z, float end_x, float end_y,
float end_z);

    float get_begin_x() const override;
    float get_begin_y() const override;
    float get_end_x() const override;
    float get_end_y() const override;
    float get_begin_z() const;
    float get_end_z() const;


    bool is_parallel(LineInSpace line) const;
    bool is_perpendicular(LineInSpace line) const;
    bool is_belongs(float x, float y, float z) const;

    string get_info();

    Vector3D convert_to_vector() const;
};
```

classes.cpp:

```
#include "classes.h"
#include <iostream>

float LineOnPlane::get_begin_x() const {
    return begin_x;
}

float LineOnPlane::get_begin_y() const {
    return begin_y;
}

float LineOnPlane::get_end_x() const {
    return end_x;
}

float LineOnPlane::get_end_y() const {
    return end_y;
}


bool LineOnPlane::is_parallel(const LineOnPlane& line) const {
    if(begin_x - end_x != 0 && line.get_begin_x() -
line.get_end_x() != 0) {
        float main_angular_coefficient = (begin_y - end_y) /
(begin_x - end_x);
        float line_angular_coefficient = (line.get_begin_y() -
line.get_end_y()) / (line.get_begin_x() - line.get_end_x());

        if (main_angular_coefficient == line_angular_coefficient)
            return true;
        else return false;
    }
    else return false;
}

bool LineOnPlane::is_perpendicular(const LineOnPlane& line) const {
    if(begin_x - end_x != 0 && line.get_begin_x() -
line.get_end_x() != 0){
        double main_angular_coefficient = (begin_y - end_y) /
(begin_x - end_x);
        double line_angular_coefficient = (line.get_begin_y() -
line.get_end_y()) / (line.get_begin_x() - line.get_end_x());
```

```cpp
        double multiply = main_angular_coefficient *
line_angular_coefficient;

        if (multiply == -1)
            return true;
        else return false;
    }
    else return false;
}

bool LineOnPlane::is_belongs(Point2D point) const {
    if(end_x - begin_x != 0 && end_y - begin_y != 0) {
        float is_x = (point.get_x() - begin_x) / (end_x - begin_x);
        float is_y = (point.get_y() - begin_y) / (end_y - begin_y);

        if (is_x == is_y)
            return true;
        else return false;
    }
    else return false;
}

LineOnPlane::LineOnPlane(float begin_x, float begin_y, float end_x,
float end_y) {
    this->begin_x = begin_x;
    this->begin_y = begin_y;
    this->end_x = end_x;
    this->end_y = end_y;
}

string LineOnPlane::get_info() {
    return "Begin x:" + to_string(int(begin_x)) + " begin y:" +
to_string(int(begin_y)) + " end x:" + to_string(int(end_x)) + " end
y:" +
            to_string(int(end_y));
}


LineInSpace::LineInSpace(float begin_x, float begin_y, float
begin_z, float end_x, float end_y, float end_z){
    this->begin_x = begin_x;
    this->begin_y = begin_y;
    this->end_x = end_x;
    this->end_y = end_y;
    this->begin_z = begin_z;
    this->end_z = end_z;
}

float LineInSpace::get_begin_x() const {
    return begin_x;
}

float LineInSpace::get_begin_y() const {
    return begin_y;
}

float LineInSpace::get_end_x() const {
    return end_x;
}

float LineInSpace::get_end_y() const {
    return end_y;
}

float LineInSpace::get_begin_z() const {
    return begin_z;
}

float LineInSpace::get_end_z() const {
    return end_z;
}

Vector3D LineInSpace::convert_to_vector() const {
    float x = end_x - begin_x;
    float y = end_y - begin_y;
    float z = end_z - begin_z;

    Vector3D vector(x,y,z);

    return vector;
}

bool LineInSpace::is_parallel(LineInSpace line) const {
    Vector3D main_vector = this->convert_to_vector();
    Vector3D line_vector = line.convert_to_vector();

    float scalar_multiply = main_vector * line_vector;

    if(scalar_multiply == 0)
```

```cpp
        return true;
    else return false;
}

bool LineInSpace::is_perpendicular(LineInSpace line) const {
    Vector3D main_vector = this->convert_to_vector();
    Vector3D line_vector = line.convert_to_vector();

    if(line_vector.get_x() != 0 && line_vector.get_y() != 0 &&
line_vector.get_z() != 0) {
        float x_ratio = main_vector.get_x() / line_vector.get_x();
        float y_ratio = main_vector.get_y() / line_vector.get_y();
        float z_ratio = main_vector.get_z() / line_vector.get_z();

        if (x_ratio == y_ratio && x_ratio == z_ratio && y_ratio ==
y_ratio)
            return true;
        else return false;
    }
    else return false;
}

bool LineInSpace::is_belongs(float x, float y, float z) const {
    if(end_x - begin_x != 0 && end_y - begin_y != 0 && end_z -
begin_z != 0) {
        float is_x = (x - begin_x) / (end_x - begin_x);
        float is_y = (y - begin_y) / (end_y - begin_y);
        float is_z = (z - begin_z) / (end_z - begin_z);

        if (is_x == is_y && is_x == is_z && is_y == is_z)
            return true;
        else return false;
    }
    else return false;
}

string LineInSpace::get_info() {
    return "Begin x:"  + to_string(int(begin_x)) + " begin y:"  +
to_string(int(begin_y)) + " begin z:"  + to_string(int(begin_z)) +
" end x:" + to_string(int(end_x)) + " end y:" +
to_string(int(end_y)) + " end z:" + to_string(int(end_z));
}


Vector3D::Vector3D(float x, float y, float z) {
    this->x = x;
    this->y = y;
    this->z = z;
}

float Vector3D::get_x() const {
    return x;
}

float Vector3D::get_y() const {
    return y;
}

float Vector3D::get_z() const {
    return z;
}

float operator*(Vector3D vec1, Vector3D vec2) {
    float multiply = vec1.get_x() * vec2.get_x() + vec1.get_y() *
vec2.get_y() + vec1.get_z() * vec2.get_z();

    return multiply;
}


Point2D::Point2D(float x, float y) {
    this->x = x;
    this->y = y;
}

float Point2D::get_x() const {
    return x;
}

float Point2D::get_y() const {
    return y;
}
```

function.cpp:

```
#ifndef INC_2LABWORK_5_FUNCTIONS_H
#define INC_2LABWORK_5_FUNCTIONS_H

#endif
#include "vector"
#include "classes.h"
using namespace std;

int capture_n();
int capture_m();
vector<LineOnPlane> generate_2d_lines(int n);
vector<LineInSpace> generate_3d_lines(int m);
Point2D capture_point();
vector<LineOnPlane> perpendicular_lines(vector<LineOnPlane>
lines_2d);
void check_if_2dlines_contains_point(vector<LineOnPlane> lines_2d,
Point2D point);
void check_if_3Dlines_contains_others(vector<LineInSpace>
lines_3d);
```

functions.cpp

```cpp
#include "functions.h"

int capture_n(){
    cout << "How many 2D lines generate: ";
    int n;
    cin >> n;

    return n;
}

int capture_m(){
    cout << "How many 3D lines generate: ";
    int m;
    cin >> m;

    return m;
}

LineOnPlane generate_random_2d_line(){
    float begin_x = rand() % 10 + 1;
    float end_x = rand() % 10 + 1;
    float begin_y = rand() % 10 + 1;
    float end_y = rand() % 10 + 1;

    LineOnPlane line(begin_x, begin_y, end_x, end_y);

    return line;
}

vector<LineOnPlane> generate_2d_lines(int n){
    vector<LineOnPlane> vec;

    for (int i = 0; i < n; ++i){
        LineOnPlane line = generate_random_2d_line();

        vec.push_back(line);
    }

    return vec;
}

LineInSpace generate_random_3d_line(){
    float begin_x = rand() % 10 + 1;
    float end_x = rand() % 10 + 1;
    float begin_y = rand() % 10 + 1;
    float end_y = rand() % 10 + 1;
    float begin_z = rand() % 10 + 1;
    float end_z = rand() % 10 + 1;

    LineInSpace line(begin_x, begin_y, begin_z, end_x, end_y,
end_z);

    return line;
}

vector<LineInSpace> generate_3d_lines(int m){
    vector<LineInSpace> vec;

    for (int i = 0; i < m; ++i){
        LineInSpace line = generate_random_3d_line();

        vec.push_back(line);
    }

    return vec;
}

Point2D capture_point(){
    cout << "Enter point's x:";
    float x;
    cin >> x;

    cout << "Enter point's y:";
    float y;
    cin >> y;

    Point2D point(x,y);

    return point;
}


vector<LineOnPlane> perpendicular_lines(vector<LineOnPlane>
lines_2d){
    LineOnPlane first_line = lines_2d[0];
    vector<LineOnPlane> perpendicular_lines;

    for (int i = 1; i < lines_2d.size(); ++i) {
        LineOnPlane line = lines_2d[1];
        if(first_line.is_perpendicular(line))
            perpendicular_lines.push_back(line);
    }

    return perpendicular_lines;
}


void check_if_2dlines_contains_point(vector<LineOnPlane> lines_2d,
Point2D point){
    if(lines_2d.empty()){
        cout << "There's no perpendicular lines to the first 2D
line\n";
    }
    else{
        for (int i = 0; i < lines_2d.size(); ++i) {
            LineOnPlane line = lines_2d[i];

            if(line.is_belongs(point))
                cout << "2D line: " << line.get_info() << "
contains point: " << point.get_x() << ":" << point.get_y() << endl;
            else
                cout << "2D line: " << line.get_info() << " doesn't
contain point: " << point.get_x() << ":" << point.get_y() << endl;
        }
    }
}

void check_if_3Dlines_contains_others(vector<LineInSpace>
lines_3d){
    for (int i = 0; i < lines_3d.size(); ++i) {
        for (int j = i + 1; j < lines_3d.size(); ++j) {
            LineInSpace line1 = lines_3d[i];
            LineInSpace line2 = lines_3d[j];

            if(line1.is_perpendicular(line2))
                printf("3D Line (%s) is perpendicular to this line:
(%s)\n", line1.get_info().c_str(), line2.get_info().c_str());
            else
                printf("3D Line (%s) isn't perpendicular to this
line: (%s)\n", line1.get_info().c_str(), line2.get_info().c_str());
        }
    }
```

```
}
```

Python

main.py

```python
from functions import *

n = capture_n()
m = capture_m()

lines_2d = generate_2d_lines(n)
lines_3d = generate_3d_lines(m)

point = capture_point()

perpendicular_line = perpendicular_lines(lines_2d)
check_if_2dlines_contains_point(perpendicular_line, point)

check_if_3Dlines_contains_others(lines_3d)
```

classes.py

```
from abc import ABC, abstractmethod


class TLine(ABC):
    @abstractmethod
    def get_begin_x(self):
        pass

    @abstractmethod
    def get_begin_y(self):
        pass

    @abstractmethod
    def get_end_x(self):
        pass

    @abstractmethod
    def get_end_y(self):
        pass


class Point2D:
    def __init__(self, x, y):
        self.__x = x
        self.__y = y

    def get_x(self):
        return self.__x

    def get_y(self):
        return self.__y


class LineOnPlane(TLine):
    def __init__(self, begin_x, begin_y, end_x, end_y):
        self.__begin_x = begin_x
        self.__begin_y = begin_y
        self.__end_x = end_x
        self.__end_y = end_y

    def get_begin_x(self):
        return self.__begin_x

    def get_begin_y(self):
        return self.__begin_y

    def get_end_x(self):
        return self.__end_x

    def get_end_y(self):
        return self.__end_y

    def is_parallel(self, line):
        if self.__begin_x - self.__end_x != 0 and
line.get_begin_x() - line.get_end_y() != 0:
            main_angular_coefficient = (self.__begin_y -
self.__end_y) / (self.__begin_x - self.__end_x)
            line_angular_coefficient = (line.get_begin_y() -
line.get_end_y) / (line.get_begin_x() - line.get_end_y())

            if main_angular_coefficient ==
line_angular_coefficient:
                return True
            else:
                return False
        else:
            return False

    def is_perpendicular(self, line):
        if self.__begin_x - self.__end_x != 0 and
line.get_begin_x() - line.get_end_y() != 0:
            main_angular_coefficient = (self.__begin_y -
self.__end_y) / (self.__begin_x - self.__end_x)
            line_angular_coefficient = (line.get_begin_y() -
line.get_end_y()) / (line.get_begin_x() - line.get_end_y())

            multiply = main_angular_coefficient *
line_angular_coefficient

            if multiply == -1:
                return True
            else:
                return False
        else:
            return False

    def is_belongs(self, point: Point2D):
        if self.__end_x - self.__begin_x != 0 and self.__end_y -
self.__begin_y != 0:
```

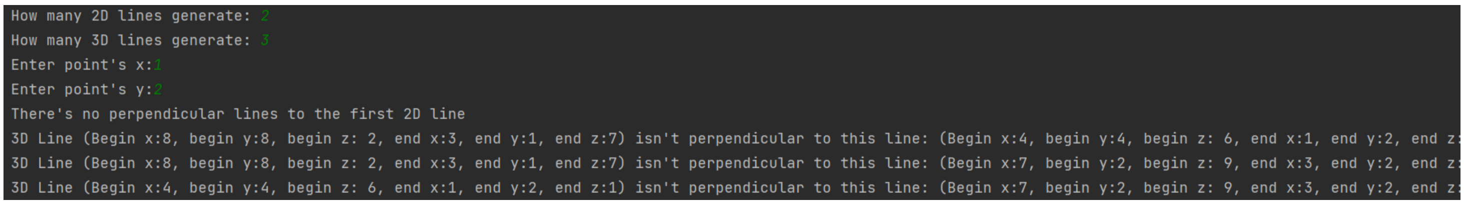
```
            is_x = (point.get_x() - self.__begin_x) / (self.__end_x
- self.__begin_x)
            is_y = (point.get_y() - self.__begin_y) / (self.__end_y
- self.__begin_y)

            if is_x == is_y:
                return True
            else:
                return False
        else:
            return False

    def get_info(self):
        return f"Begin x:{self.__begin_x}, begin
y:{self.__begin_y}, end x:{self.__end_x}, end y:{self.__end_y}"


class Vector3D:
    def __init__(self, x, y, z):
        self.__x = x
        self.__y = y
        self.__z = z

    def get_x(self):
        return self.__x

    def get_y(self):
        return self.__y

    def get_z(self):
        return self.__z

    def __mul__(self, other):
        multiply = self.get_x() * other.get_x() + self.get_y() *
other.get_y() + self.get_z() * other.get_z()

        return multiply


class LineInSpace(TLine):
    def __init__(self, begin_x, begin_y, begin_z, end_x, end_y,
end_z):
        self.__begin_x = begin_x
        self.__begin_y = begin_y
        self.__begin_z = begin_z
```

```python
        self.__end_x = end_x
        self.__end_y = end_y
        self.__end_z = end_z

    def get_begin_x(self):
        return self.__begin_x

    def get_begin_y(self):
        return self.__begin_y

    def get_begin_z(self):
        return self.__begin_z

    def get_end_x(self):
        return self.__end_x

    def get_end_y(self):
        return self.__end_y

    def get_end_z(self):
        return self.__end_z

    def convert_to_vector(self):
        x = self.__end_x - self.__begin_x
        y = self.__end_y - self.__begin_y
        z = self.__end_z - self.__begin_z

        vector = Vector3D(x, y, z)

        return vector

    def is_parallel(self, line):
        main_vector = self.convert_to_vector()
        line_vector = line.convert_to_vector()

        scalar_multiply = main_vector * line_vector

        if scalar_multiply == 0:
            return True
        else:
            return False

    def is_perpendicular(self, line):
        main_vector = self.convert_to_vector()
        line_vector = line.convert_to_vector()
        if line_vector.get_x() != 0 and line_vector.get_y() != 0
and line_vector.get_z() != 0:
            x_ratio = main_vector.get_x() / line_vector.get_x()
            y_ratio = main_vector.get_y() / line_vector.get_y()
            z_ratio = main_vector.get_z() / line_vector.get_z()

            if x_ratio == y_ratio and x_ratio == z_ratio and
y_ratio == y_ratio:
                return True
            else:
                return False
        else:
            return False

    def is_belongs(self, x, y, z):
        if self.__end_x - self.__begin_x != 0 and self.__end_y -
self.__begin_y != 0 and self.__end_z - self.__begin_z != 0:
            is_x = (x - self.__begin_x) / (self.__end_x -
self.__begin_x)
            is_y = (y - self.__begin_y) / (self.__end_y -
self.__begin_y)
            is_z = (z - self.__begin_z) / (self.__end_z -
self.__begin_z)

            if is_x == is_y and is_x == is_z and is_y == is_z:
                return True
            else:
                return False
        else:
            return False

    def get_info(self):
        return f"Begin x:{self.__begin_x}, begin
y:{self.__begin_y}, begin z: {self.__begin_z}, end
x:{self.__end_x}, end y:{self.__end_y}, end z:{self.__end_z}"
```

function.py

```python
from classes import *
import random


def capture_n():
    n = int(input("How many 2D lines generate: "))

    return n


def capture_m():
    m = int(input("How many 3D lines generate: "))

    return m


def generate_random_2d_line():
    begin_x = random.randint(1,10)
    end_x = random.randint(1,10)
    begin_y = random.randint(1,10)
    end_y = random.randint(1,10)

    line = LineOnPlane(begin_x, begin_y, end_x, end_y)

    return line


def generate_2d_lines(n: int):
    lines = []

    for i in range(n):
        line = generate_random_2d_line()
        lines.append(line)

    return lines


def generate_random_3d_line():
    begin_x = random.randint(1, 10)
    end_x = random.randint(1, 10)
    begin_y = random.randint(1, 10)
    end_y = random.randint(1, 10)
    begin_z = random.randint(1, 10)
    end_z = random.randint(1, 10)
    line = LineInSpace(begin_x, begin_y, begin_z, end_x, end_y,
end_z)

    return line


def generate_3d_lines(m: int):
    lines = []

    for i in range(m):
        line = generate_random_3d_line()

        lines.append(line)

    return lines


def capture_point():
    x = int(input("Enter point's x:"))
    y = int(input("Enter point's y:"))

    point = Point2D(x,y)

    return point


def perpendicular_lines(lines_2d: list[LineOnPlane]):
    first_line = lines_2d[0]
    perpendicular_lines = []

    for i in range(len(lines_2d)):
        line = lines_2d[1]
        if first_line.is_perpendicular(line):
            perpendicular_lines.append(line)

    return perpendicular_lines


def check_if_2dlines_contains_point(lines_2d: list[LineOnPlane],
point: Point2D):
    if len(lines_2d) == 0:
        print("There's no perpendicular lines to the first 2D
line")
    else:
        for i in range(len(lines_2d)):
            line = lines_2d[i]
            if line.is_belongs(point):
                print(f"2D line: {line.get_info()} contains point:
{point.get_x()}:{point.get_y()}")
            else:
                print(f"2D line: {line.get_info()} doesn't contain
point: {point.get_x()}:{point.get_y()}")


def check_if_3Dlines_contains_others(lines_3d: list[LineInSpace]):
    for i in range(len(lines_3d)):
        for j in range(i+1, len(lines_3d)):
            line1 = lines_3d[i]
            line2 = lines_3d[j]

            if line1.is_perpendicular(line2):
                print(f"3D Line ({line1.get_info()}) is
perpendicular to this line: ({line2.get_info()})")
            else:
                print(f"3D Line ({line1.get_info()}) isn't
perpendicular to this line: ({line2.get_info()})")
```

Результат виконання програми



```
How many 2D lines generate: 2
How many 3D lines generate: 3
Enter point's x:1
Enter point's y:2
There's no perpendicular lines to the first 2D line
3D Line (Begin x:8, begin y:8, begin z: 2, end x:3, end y:1, end z:7) isn't perpendicular to this line: (Begin x:4, begin y:4, begin z: 6, end x:1, end y:2, end z:
3D Line (Begin x:8, begin y:8, begin z: 2, end x:3, end y:1, end z:7) isn't perpendicular to this line: (Begin x:7, begin y:2, begin z: 9, end x:3, end y:2, end z:
3D Line (Begin x:4, begin y:4, begin z: 6, end x:1, end y:2, end z:1) isn't perpendicular to this line: (Begin x:7, begin y:2, begin z: 9, end x:3, end y:2, end z:
```