# 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»

Варіант 13

Виконав студент Котков Тимур Максимович

(шифр, прізвище, ім'я, по батькові)

Перевірив Вітковська Ірина Іванівна

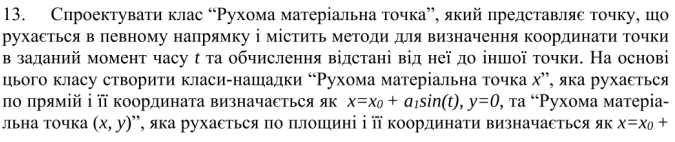
( прізвище, ім'я, по батькові)

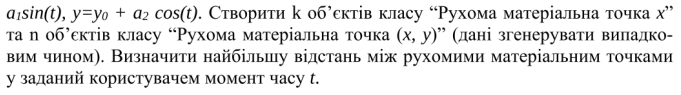
Київ 2022

**Лабораторна робота №5**

**Успадкування та поліморфізм**

**Мета –** визначити механізми створення класів та об’єктів.

**Індивідуальне завдання:**

****

**Постановка задачі**

Розробимо клас MaterialPoint з конструкторами, що приймають як параметри число t. Реалізуємо успадкування деяких функцій, деякі залишимо статичними. Проводимо операції зі створенням, додаванням і порівнянням згідно умови, виводимо результат у консоль.

**Код на С++ :**

**Header.h :**

#pragma once

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include <iostream>

#include <vector>

#include <math.h> // cos, sin

using namespace std;

class materialPoint {

protected:

int a1 = rand() % 5 + 1, a2 = rand() % 5 + 1, x0 = rand() % 5 + 1, y0 = rand() % 5 + 1;

double x, y, t;

virtual double findX() = 0;

virtual double findY() = 0;

virtual double getDistance(materialPoint\* mp) = 0;

materialPoint(double t);

public:

virtual double getX() = 0;

virtual double getY() = 0;

virtual void getCoordinates() = 0;

};

class materialPointX : public materialPoint {

private:

double findX() override;

double findY() override;

double x = findX(), y = findY();

public:

double getX() override {

return x;

}

double getY() override {

return y;

}

void getCoordinates() override {

cout << "(x;y) = " << x << "; " << y << endl;

}

double getDistance(materialPoint\* mp) override;

materialPointX(float t) : materialPoint(t) {};

};

class materialPointXY : public materialPoint {

private:

double findX() override;

double findY() override;

double x = findX(), y = findY();

public:

double getX() override {

return x;

}

double getY() override {

return y;

}

void getCoordinates() override {

cout << "(x;y) = " << x << "; " << y << endl;

}

double getDistance(materialPoint\* mp) override;

materialPointXY(float t) : materialPoint(t) {};

};

**Header.h :**

#pragma once

#include <iostream>

#include <vector>

using namespace std;

int findMax(vector<double> distances);

**Source.cpp**

#include "Header.h"

#include "Number.h"

int vectorToNum(Number& num) {

int n = 0;

for (int i = 0; i < num.numByDigit.size(); i++)

{

n += num.numByDigit[i] \* pow(10, num.numByDigit.size() - i - 1);

}

return n;

}

**MaterialPoint.cpp:**

#include "materialPoint.h"

materialPoint::materialPoint(double t) {

this->t = t;

}

double materialPointX::findX(){

return x0 + a1 \* sin(t);

}

double materialPointX::findY() {

return 0;

}

double materialPointXY::findX() {

return x0 + a1 \* sin(t);

}

double materialPointXY::findY() {

return y0 + a2 \* cos(t);

}

double materialPointX::getDistance(materialPoint \*mp) {

float x2 = mp->getX();

float y2 = mp->getY();

return sqrt(pow((this->x - x2), 2) + pow((this->y - y2), 2));

}

double materialPointXY::getDistance(materialPoint \*mp) {

float x2 = mp->getX();

float y2 = mp->getY();

return sqrt(pow((this->x - x2), 2) + pow((this->y - y2), 2));

}

Source.cpp

#include "materialPoint.h"

int findMax(vector<double> distances) {

int ind = 0;

for (int i = 0; i < distances.size(); i++)

{

if (distances[ind] <= distances[i]) {

ind = i;

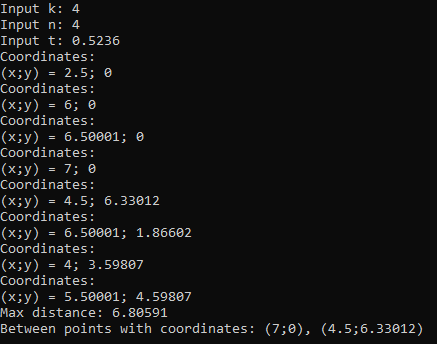
}

}

return ind;

}

**Скріншот результатів програми на C++ :**

****

**Код на Python:**

**material\_point.py:**

*from* abc *import* ABCMeta, abstractmethod, ABC  
*import* random  
*import* math  
  
  
*class* MaterialPoint(metaclass=ABCMeta):  
 *def* \_\_init\_\_(*self*, t):  
 *self*.\_a1, *self*.\_a2, *self*.\_x0, *self*.\_y0 = random.randint(0, 5), random.randint(0, 5), \  
 random.randint(0, 5), random.randint(0, 5)  
 *self*.\_t = t  
 *self*.\_x, *self*.\_y = *self*.\_find\_x(), *self*.\_find\_y()  
  
 *def* \_\_str\_\_(*self*):  
 *return* f'(x;y) = ({*self*.\_x}; {*self*.\_y})'  
  
 @abstractmethod  
 *def* \_find\_x(*self*): *pass* @abstractmethod  
 *def* \_find\_y(*self*): *pass* @abstractmethod  
 *def* get\_x(*self*): *pass* @abstractmethod  
 *def* get\_y(*self*): *pass* @abstractmethod  
 *def* get\_distance(*self*, ox, oy): *pass*

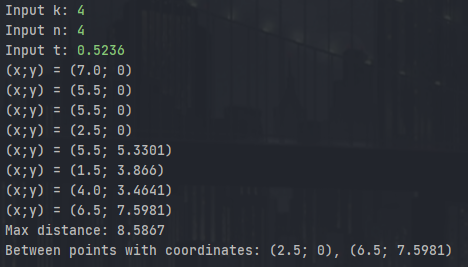
*class* MaterialPointX(MaterialPoint, ABC):  
 *def* \_\_init\_\_(*self*, t):  
 super().\_\_init\_\_(t)  
  
 *def* \_find\_x(*self*):  
 *return* round(*self*.\_x0 + *self*.\_a1 \* math.sin(*self*.\_t), 4)  
  
 *def* get\_x(*self*):  
 *return self*.\_x  
  
 *def* \_find\_y(*self*):  
 *return* 0  
  
 *def* get\_y(*self*):  
 *return self*.\_y  
  
 *def* get\_distance(*self*, ox, oy):  
 *return* round(math.sqrt((*self*.\_x - ox) \*\* 2 + (*self*.\_y - oy) \*\* 2), 4)  
  
  
*class* MaterialPointXY(MaterialPoint, ABC):  
 *def* \_\_init\_\_(*self*, t):  
 super().\_\_init\_\_(t)  
  
 *def* \_find\_x(*self*):  
 *return* round(*self*.\_x0 + *self*.\_a1 \* math.sin(*self*.\_t), 4)  
  
 *def* get\_x(*self*):  
 *return self*.\_x  
  
 *def* \_find\_y(*self*):  
 *return* round(*self*.\_y0 + *self*.\_a2 \* math.cos(*self*.\_t), 4)  
  
 *def* get\_y(*self*):  
 *return self*.\_y  
  
 *def* get\_distance(*self*, ox, oy):  
 *return* round(math.sqrt((*self*.\_x - ox) \*\* 2 + (*self*.\_y - oy) \*\* 2), 4)

**lab5.py:**

*from* material\_point *import* MaterialPointX, MaterialPointXY  
  
k = int(input("Input k: "))  
n = int(input("Input n: "))  
t = float(input("Input t: "))  
  
coords = []  
distances = []  
xs = []  
xys = []  
  
*for* i *in* range(k):  
 cl = MaterialPointX(t)  
 xs.append(cl)  
 print(cl)  
 *for* j *in* range(i):  
 dist = cl.get\_distance(xs[j].get\_x(), xs[j].get\_y())  
 coordinates = [cl.get\_x(), cl.get\_y(), xs[j].get\_x(), xs[j].get\_y()]  
 coords.append(coordinates)  
 distances.append(dist)

*for* i *in* range(n):  
 cl = MaterialPointXY(t)  
 xys.append(cl)  
 print(cl)  
 *for* j *in* range(i):  
 dist = cl.get\_distance(xys[j].get\_x(), xys[j].get\_y())  
 coordinates = [cl.get\_x(), cl.get\_y(), xys[j].get\_x(), xys[j].get\_y()]  
 coords.append(coordinates)  
 distances.append(dist)  
  
*for* i *in* range(k):  
 *for* j *in* range(n):  
 dist = xs[i].get\_distance(xys[j].get\_x(), xys[j].get\_y())  
 coordinates = [xs[i].get\_x(), xs[i].get\_y(), xys[j].get\_x(), xys[j].get\_y()]  
 coords.append(coordinates)  
 distances.append(dist)  
  
ind = distances.index(max(distances))  
print(f'Max distance: {distances[ind]}')  
print(f'Between points with coordinates: ({coords[ind][0]}; {coords[ind][1]}), ({coords[ind][2]}; {coords[ind][3]})')

**Результат виконання коду на Python:**

****

**Висновок:**

Під час лабораторної роботи ми навчилися поліморфізму. Також ознайомилися з override-функціями у класі. Оскільки програма працює згідно з умовою, то можна сказати, що вона працює правильно.