# 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 29

Виконав студент ІП-11 Тарасьонок Дмитро Євгенович

(шифр, прізвище, ім'я, по батькові)

Перевірив Вітковська Ірина Іванівна

( прізвище, ім'я, по батькові)

Київ 2022

Мета: вивчити механізми створення і використання класів та об’єктів

Умова задачі: Спроектувати клас "Особа", який містить ПІБ, дата народження і методи визначення її віку та обчислення місячного заробітку. На основі цього класу створити класи-нащадки "Студент", який містить додатково номер академічної групи студента, середній рейтинговий бал за результатами останньої сесії, статус отримання стипендії у новому семетрі (підвищена, звичайна, немає), та "Викладач", який містить додатково назви дисциплін, які викладає даний викладач, планову кількість годин, які повинен провести по кожній дисципліні, за місяць. Створити n студентів і m викладачів. Для студентів розрахувати місячної стипендії (підвищена, якщо рейтинговий бал більше 95, і звичайна - якщо середній рейтинговий бал більше 85, але менше 95), для викладачів - місячну заробітну плату. Визначити вік викладача, що має найбільшу місячну заробітну плату.

Постановка задачі: для виконання поставленої задачі нам треба створити базовий клас Person, що буде відповідати за задання імені та дати народження, буде обраховувати вік особи та матиме одну чисто віртуальну функцію – функцію обчислення заробітку, оскільки ця функція буде абсолютно різною у викладача та студента. Статус стипендії пропонується не зберігати як поле, а просто обраховувати щоразу, коли це треба. Для обрахунку зарплати викладача пропонується взяти базову оплату за одну годину роботи та просумувати добутки цієї базової оплати на планову кількість годин для кожної дисципліни, що цей викладач викладає. Генерацію випадкових осіб пропонується виконувати, використовуючи заданий набір імен, дату народження – шукати дату в проміжку з 1950 по 2004 р.н., для всіх інших полів, що використовуються, обрати такий самий підхід, як і з іменем у класі особи. Викладача з найбільшою зарплатою можна визначити повним проходом по масиву викладачів із порівнянням зарплат викладачів.

Програма мовою C++:

Файл main.cpp

/\*

\* 1 курс, 2 семестр, Основи програмування, лабораторна робота №5.

\*

\* Варіант 29

\* Спроектувати клас "Особа", який містить ПІБ, дата народження і методи

\* визначення її віку та обчислення місячного заробітку. На основі цього класу

\* створити класи-нащадки "Студент", який містить додатково номер академічної

\* групи студента, середній рейтинговий бал за результатами останньої сесії,

\* статус отримання стипендії у новому семетрі (підвищена, звичайна, немає), та

\* "Викладач", який містить додатково назви дисциплін, які викладає даний

\* виклада, планову кількість годин, які повинен провести по кожній дисципліні,

\* за місяць. Створити n студентів і m викладачів. Для студентів розрахувати

\* розмір місячної стипендії (підвищена, якщо рейтинговий бал більше 95, і

\* звичайна - якщо середній рейтинговий бал більше 85, але менше 95), для

\* викладачів - місячну заробітну плату. Визначити вік викладача, що має

\* найбільшу місячну заробітну плату.

\*/

#include <locale>

#include <Windows.h>

#include "Functions.h"

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

locale::global(locale("rus\_rus.1251"));

srand(time(nullptr));

size\_t n, m;

cout << "Скільки студентів згенерувати?: ";

cin >> n;

cout << "Скільки викладачів згенерувати?: ";

cin >> m;

Student\* rgStudents = generateStudents(n);

Teacher\* rgTeachers = generateTeachers(m);

cout << endl << "Згенеровані студенти:" << endl;

outputStudents(rgStudents, n);

cout << "Згенеровані викладачі:" << endl;

outputTeachers(rgTeachers, m);

Teacher teacherWithHighestSalary = getTeacherWithHighestSalary(rgTeachers, m);

cout << "Викладач із найбільною зарплатою:" << endl << teacherWithHighestSalary << endl;

system("pause");

return 0;

}

Файл Discipline.h

#pragma once

#include <string>

#include <iostream>

using std::string;

using std::ostream;

using std::endl;

struct Discipline

{

Discipline(const string& sName, double nNumberOfHours) : m\_sName{sName}, m\_nNumberOfHours{nNumberOfHours} {}

Discipline();

string m\_sName;

double m\_nNumberOfHours;

};

ostream& operator << (ostream& os, const Discipline& a\_discipline);

Файл Discipline.cpp

#include "Discipline.h"

Discipline::Discipline()

{

string rgDisciplinesNames[35]{

"Комп'ютерна дискретна математика",

"Математичний аналіз",

"Лінійна алгебра та аналітична геометрія",

"Українська мова за професійним спрямуванням",

"Екологічна безпека та цивільний захист",

"Історія науки і техніки",

"Основи здорового способу життя",

"Іноземна мова",

"Теорія ймовірностей",

"Групова динаміка та комунікації",

"Філософські основи наукового пізнання",

"Права і свободи людини",

"Економіка IT - індустрії та підприємництво",

"Іноземна мова професійного спрямування",

"Фізика",

"Алгоритми та структури даних",

"Основи програмування",

"Основи компютерних систем і мереж",

"Бази даних",

"Курсова робота з баз даних",

"Компоненти програмної інженерії",

"Безпека програмного забезпечення",

"Проектування алгоритмів",

"Системне програмне забезпечення",

"Програмування веб - застосувань",

"Ймовірністні моделі та статистичне оцінювання в інформаційних системах",

"Аналіз даних в інформаційних системах",

"Курсова робота з аналізу даних в інформаційних системах",

"Мультипарадигмене програмування",

"Програмування інтернет речей",

"Методи оптимізації та прийняття рішень",

"Технології паралельних обчислень",

"Програмування інтелектуальних інформаційних систем",

"Моделювання систем",

"Інформаційні системи та технології"

};

m\_sName = rgDisciplinesNames[rand() % 35];

m\_nNumberOfHours = 2 + rand() % 11;

}

ostream& operator << (ostream& os, const Discipline& a\_discipline)

{

return os << "Назва дисципліни: " << a\_discipline.m\_sName << endl

<< "Кількість годин: " << a\_discipline.m\_nNumberOfHours;

}

Файл Functions.h

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "Student.h"

#include "Teacher.h"

Student\* generateStudents(size\_t a\_cCounter);

Teacher\* generateTeachers(size\_t a\_cCounter);

void outputStudents(const Student\* students, size\_t size);

void outputTeachers(const Teacher\* teachers, size\_t size);

Teacher getTeacherWithHighestSalary(Teacher\* teachers, size\_t size);

Файл Functions.cpp

#include "Functions.h"

Student\* generateStudents(size\_t a\_cCounter)

{

Student\* arr = new Student[a\_cCounter];

for (size\_t i = 0; i < a\_cCounter; ++i)

{

arr[i] = Student();

}

return arr;

}

Teacher\* generateTeachers(size\_t a\_cCounter)

{

Teacher\* arr = new Teacher[a\_cCounter];

for (size\_t i = 0; i < a\_cCounter; ++i)

{

arr[i] = Teacher();

}

return arr;

}

void outputStudents(const Student\* students, size\_t size)

{

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

{

cout << students[i] << endl << endl;

}

}

void outputTeachers(const Teacher\* teachers, size\_t size)

{

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

{

cout << teachers[i] << endl << endl;

}

}

Teacher getTeacherWithHighestSalary(Teacher\* teachers, size\_t size)

{

size\_t iHighestIndex = 0;

double dbHighestSalary = teachers[0].getSalary();

for (int i = 1; i < size; ++i)

{

if (dbHighestSalary < teachers[i].getSalary())

{

iHighestIndex = i;

dbHighestSalary = teachers[i].getSalary();

}

}

return teachers[iHighestIndex];

}

Файл Person.h

#pragma once

#include "TDate.h"

using std::string;

class Person

{

public:

Person(const string& name, const TDate& dtBirthday) :m\_sName{ name }, m\_dtBirthday{ dtBirthday } {}

Person();

string getName() const { return m\_sName; }

TDate getBirthday() const { return m\_dtBirthday; }

int getAge() const;

virtual double getSalary() const = 0;

private:

string m\_sName;

TDate m\_dtBirthday;

};

Файл Person.cpp

#include "Person.h"

int Person::getAge() const

{

TDate dtCurrent = TDate::getCurrentDate();

int nYears;

int aDaysInMonth[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

nYears = dtCurrent.year - m\_dtBirthday.year;

if (dtCurrent.month < m\_dtBirthday.month || (dtCurrent.month == m\_dtBirthday.month && dtCurrent.day < m\_dtBirthday.day))

{

--nYears;

}

return nYears;

}

Person::Person()

{

string rgLastNames[31]{

"Алєксєєв", "Боровков", "Веремчук", "Гіжицький",

"Головатюк", "Головня", "Гуськов", "Демиденко",

"Друзенко", "Дякунчак", "Звєрєв", "Калашніков",

"Кашка", "Книш", "Кузьменков", "Куценко",

"Лесів", "Лисенко", "Лошак", "Ляля", "Негрієнко",

"Панченко", "Прищепа", "Рябов", "Савенко",

"Сідак", "Тарасьнок", "Тихонов", "Трикош",

"Шамков", "Ющенко"

};

string rgFirstNames[16]{

"Андрій", "Іван", "Ігор", "Даніїл", "Владислав",

"Олександр", "Кирило", "Олексій", "Микола",

"Максим", "Дмитро", "Артемій", "Віктор",

"Сергій", "Юрій", "Федір"

};

string rgPatronymics[20]{

"Олександрович", "Ігорович", "Денисович",

"Ростиславович", "Михайлович", "Олексійович",

"Юрійович", "Віталійович", "Миколайович",

"Євгенович", "Сергійович", "Олегович",

"Ілліч", "Іванович", "Костянтинович",

"Станіславович", "Андрійович",

"Володимирович", "Дмитрович",

"Вікторович"

};

m\_sName = rgLastNames[rand() % 31] + ' ' + rgFirstNames[rand() % 16] + ' ' + rgPatronymics[rand() % 20];

m\_dtBirthday = TDate();

}

Файл Scholarship.h

#pragma once

#include <iostream>

using std::ostream;

using std::istream;

enum class Scholarship

{

None, Default, Increased

};

ostream& operator << (ostream&, const Scholarship&);

Файл Scholarship.cpp

#include "Scholarship.h"

ostream& operator << (ostream& os, const Scholarship& a\_scholarship)

{

if (a\_scholarship == Scholarship::Increased)

{

os << "Підвищена";

}

else if (a\_scholarship == Scholarship::Default)

{

os << "Звичайна";

}

else

{

os << "Відсутня";

}

return os;

}

Файл Student.h

#pragma once

#include "Person.h"

#include "Scholarship.h"

class Student : public Person

{

public:

Student(const string&, const TDate&, const string&, double);

Student();

Scholarship getScholarship() const;

double getSalary() const;

string getGroupName() const { return m\_sGroupName; }

double getGradePointAverage() const { return m\_dbGradePointAverage; }

private:

string m\_sGroupName;

double m\_dbGradePointAverage;

};

ostream& operator << (ostream&, const Student&);

Файл Student.cpp

#include "Student.h"

Student::Student(const string& name, const TDate& dtBirthday, const string& sGroupName, double dbGradePointAverage)

:Person(name, dtBirthday), m\_sGroupName{ sGroupName }, m\_dbGradePointAverage{ dbGradePointAverage }

{}

Scholarship Student::getScholarship() const

{

Scholarship resultScholarship;

if (95 <= m\_dbGradePointAverage)

{

resultScholarship = Scholarship::Increased;

}

else if (85 <= m\_dbGradePointAverage)

{

resultScholarship = Scholarship::Default;

}

else

{

resultScholarship = Scholarship::None;

}

return resultScholarship;

}

double Student::getSalary() const

{

Scholarship scholarship = getScholarship();

double nScholarshipSum;

if (scholarship == Scholarship::Default)

{

nScholarshipSum = 2000;

}

else if (scholarship == Scholarship::Increased)

{

nScholarshipSum = 2550;

}

else

{

nScholarshipSum = 0;

}

return nScholarshipSum;

}

ostream& operator << (ostream& os, const Student& a\_student)

{

os << "ПІБ студента: " << a\_student.getName() << endl

<< "Дата народження: " << a\_student.getBirthday() << " (вік: " << a\_student.getAge() << ')' << endl

<< "Номер академічної групи: " << a\_student.getGroupName() << endl

<< "Середній рейтинговий бал: " << a\_student.getGradePointAverage() << endl

<< "Стипендія: " << a\_student.getScholarship() << endl

<< "Місячний заробіток: " << a\_student.getSalary() << " грн.";

return os;

}

Student::Student()

{

string rgGroupPrefixes[8] = {

"ІА", "ІК", "ІС", "ІП", "ІТ", "ІО", "ІМ", "ІВ"

};

m\_sGroupName = rgGroupPrefixes[rand() % 8] + '-' + to\_string(11 + rand() % 5);

m\_dbGradePointAverage = static\_cast<double>(6000 + rand() % 4001) / 100;

}

Файл TDate.h

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <ctime>

#include <iostream>

#include <string>

using namespace std;

struct TDate

{

TDate(int a\_nDay, int a\_nMonth, int a\_nYear);

TDate();

static TDate getCurrentDate();

int day;

int month;

int year;

};

ostream& operator << (ostream&, const TDate&);

Файл TDate.cpp

#include "TDate.h"

TDate::TDate(int a\_nDay, int a\_nMonth, int a\_nYear)

:day{ a\_nDay }, month{ a\_nMonth }, year{ a\_nYear }

{

time\_t dtCurrentTimeT = time(nullptr);

tm\* dtNow = localtime(&dtCurrentTimeT);

int now\_year = dtNow->tm\_year + 1900;

int now\_month = dtNow->tm\_mon + 1;

int now\_day = dtNow->tm\_mday;

int aDaysInMonth[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

if (now\_year < this->year)

{

this->year = now\_year;

}

else if (this->year < 1900)

{

this->year = 1900;

}

if (12 < this->month)

{

this->month = 12;

}

else if (this->month < 1)

{

this->month = 1;

}

if (this->year == now\_year && now\_month < this->month)

{

this->month = now\_month;

}

if (aDaysInMonth[this->month - 1] < this->day)

{

this->day = aDaysInMonth[this->month - 1];

}

else if (this->day < 1)

{

this->day = 1;

}

if (this->year == now\_year && this->month == now\_month && now\_day < this->day)

{

this->day = now\_day;

}

}

TDate TDate::getCurrentDate()

{

time\_t dtCurrentTimeT = time(nullptr);

tm\* dtNow = localtime(&dtCurrentTimeT);

return TDate(dtNow->tm\_mday, dtNow->tm\_mon + 1, dtNow->tm\_year + 1900);

}

ostream& operator << (ostream& os, const TDate& a\_dt)

{

if (a\_dt.day < 10)

{

os << 0;

}

os << a\_dt.day << '.';

if (a\_dt.month < 10)

{

os << 0;

}

os << a\_dt.month << '.' << to\_string(a\_dt.year);

return os;

}

TDate::TDate()

{

int nDay, nMonth, nYear;

int aDaysInMonth[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

nYear = 1950 + rand() % 55;

nMonth = 1 + rand() % 12;

nDay = 1 + rand() % aDaysInMonth[nMonth];

day = nDay;

month = nMonth;

year = nYear;

}

Файл Teacher.h

#pragma once

#include "Person.h"

#include "Discipline.h"

class Teacher : public Person

{

public:

Teacher(const string& name, const TDate& dtBirthday);

Teacher();

double getSalary() const;

Discipline\* getDisciplines() const { return m\_rgDisciplines; }

size\_t getDisciplinesCount() const { return m\_cDisciplinesCount; }

Teacher& addDiscipline(const Discipline&);

private:

Discipline\* m\_rgDisciplines;

size\_t m\_cDisciplinesCount;

};

ostream& operator << (ostream&, const Teacher&);

Файл Teacher.cpp

#include "Teacher.h"

Teacher::Teacher(const string& a\_sName, const TDate& a\_dt)

: Person(a\_sName, a\_dt), m\_rgDisciplines{ nullptr }, m\_cDisciplinesCount{0}

{}

Teacher::Teacher()

: m\_rgDisciplines{ nullptr }, m\_cDisciplinesCount { 0 }

{

int nDisciplinesGoal = 1 + rand() % 3;

while (m\_cDisciplinesCount != nDisciplinesGoal)

{

Discipline randomDiscipline;

bool bIsInArr = false;

for (size\_t i = 0; i < m\_cDisciplinesCount; ++i)

{

if (m\_rgDisciplines[i].m\_sName == randomDiscipline.m\_sName)

{

bIsInArr = true;

}

}

if (!bIsInArr)

{

addDiscipline(randomDiscipline);

}

}

}

double Teacher::getSalary() const

{

int nSalaryBase = 500;

double nTotalSalary = 0;

for (size\_t i = 0; i < getDisciplinesCount(); ++i)

{

nTotalSalary += m\_rgDisciplines[i].m\_nNumberOfHours \* nSalaryBase;

}

return nTotalSalary;

}

Teacher& Teacher::addDiscipline(const Discipline& a\_discipline)

{

m\_cDisciplinesCount += 1;

Discipline\* newDisciplines = new Discipline[m\_cDisciplinesCount];

for (int i = 0; i < m\_cDisciplinesCount - 1; ++i)

{

newDisciplines[i] = m\_rgDisciplines[i];

}

if (m\_rgDisciplines != nullptr)

{

delete[] m\_rgDisciplines;

}

newDisciplines[m\_cDisciplinesCount - 1] = a\_discipline;

m\_rgDisciplines = newDisciplines;

return \*this;

}

ostream& operator << (ostream& os, const Teacher& a\_teacher)

{

os << "ПІБ викладача: " << a\_teacher.getName() << endl

<< "Дата народження: " << a\_teacher.getBirthday() << " (вік: " << a\_teacher.getAge() << ')' << endl

<< "Місячний заробіток: " << a\_teacher.getSalary() << " грн." << endl

<< "Дисципліни викладача:" << endl;

Discipline\* rgTeacherDisciplines = a\_teacher.getDisciplines();

for (size\_t i = 0; i < a\_teacher.getDisciplinesCount(); ++i)

{

os << rgTeacherDisciplines[i];

if (i != a\_teacher.getDisciplinesCount() - 1)

{

os << endl;

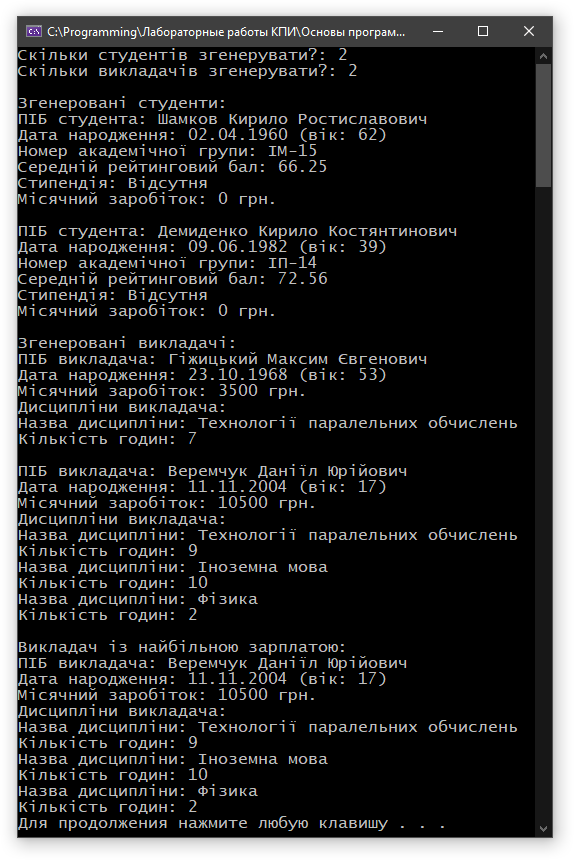
}

}

return os;

}

Виконання коду мовою C++:



Програма мовою Python

Файл main.py

*"""  
1 курс, 2 семестр, Основи програмування, лабораторна робота №5.  
  
Варіант 29  
Спроектувати клас "Особа", який містить ПІБ, дата народження і методи  
визначення її віку та обчислення місячного заробітку. На основі цього класу  
створити класи-нащадки "Студент", який містить додатково номер академічної  
групи студента, середній рейтинговий бал за результатами останньої сесії,  
статус отримання стипендії у новому семетрі (підвищена, звичайна, немає), та  
"Викладач", який містить додатково назви дисциплін, які викладає даний  
виклада, планову кількість годин, які повинен провести по кожній дисципліні,  
за місяць. Створити n студентів і m викладачів. Для студентів розрахувати  
розмір місячної стипендії (підвищена, якщо рейтинговий бал більше 95, і  
звичайна - якщо середній рейтинговий бал більше 85, але менше 95), для  
викладачів - місячну заробітну плату. Втзначити вік викладача, що має  
найбільшу місячну заробітну плату.  
"""  
from* Person *import* Person  
*from* Functions *import* generate\_students, generate\_teachers, \  
 get\_teacher\_with\_highest\_salary  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 n = int(input('Скільки студентів згенерувати?: '))  
 m = int(input('Скільки викладачів згенерувати?: '))  
  
 students = generate\_students(n)  
 teachers = generate\_teachers(m)  
  
 print(f'\nЗгенеровані студенти:')  
 print('\n\n'.join([str(student) *for* student *in* students]))  
  
 print('\nЗгенеровані студенти:')  
 print('\n'.join([str(teacher) *for* teacher *in* teachers]))  
  
 teacher\_with\_highest\_salary = get\_teacher\_with\_highest\_salary(teachers)  
  
 print(  
 'Викладач із найбільшою зарплатою:',  
 teacher\_with\_highest\_salary,  
 sep='\n'  
 )

Файл Functions/\_\_init\_\_.py

*from* Student *import* Student  
*from* Teacher *import* Teacher  
*import* typing  
  
*def* generate\_students(counter: int) -> typing.List[Student]:  
 students = []  
  
 *for* i *in* range(counter):  
 students.append(Student())  
  
 *return* students  
  
  
*def* generate\_teachers(counter: int) -> typing.List[Teacher]:  
 teachers = []  
  
 *for* i *in* range(counter):  
 teachers.append(Teacher())  
  
 *return* teachers  
  
  
*def* get\_teacher\_with\_highest\_salary(teachers: typing.List[Teacher]):  
 highest\_index = 0  
 highest\_salary = teachers[0].get\_salary()  
  
 *for* index, teacher *in* enumerate(teachers[1:], 1):  
 *if* highest\_salary < teacher.get\_salary():  
 highest\_index = index  
 highest\_salary = teacher.get\_salary()  
  
 *return* teachers[highest\_index]

Файл Person/\_\_init\_\_.py

*import* datetime  
*import* typing  
*import* random  
*from* abc *import* abstractmethod  
  
  
*class* Person(object):  
 *def* \_\_init\_\_(  
 *self*,  
 name=*None*,  
 birthday=*None*,  
 ):  
 *if* name *is None*:  
 last\_names = (  
 "Алєксєєв", "Боровков", "Веремчук", "Гіжицький",  
 "Головатюк", "Головня", "Гуськов", "Демиденко",  
 "Друзенко", "Дякунчак", "Звєрєв", "Калашніков",  
 "Кашка", "Книш", "Кузьменков", "Куценко",  
 "Лесів", "Лисенко", "Лошак", "Ляля", "Негрієнко",  
 "Панченко", "Прищепа", "Рябов", "Савенко",  
 "Сідак", "Тарасьнок", "Тихонов", "Трикош",  
 "Шамков", "Ющенко"  
 )  
  
 first\_names = (  
 "Андрій", "Іван", "Ігор", "Даніїл", "Владислав",  
 "Олександр", "Кирило", "Олексій", "Микола",  
 "Максим", "Дмитро", "Артемій", "Віктор",  
 "Сергій", "Юрій", "Федір"  
 )  
  
 patronymics = (  
 "Олександрович", "Ігорович", "Денисович",  
 "Ростиславович", "Михайлович", "Олексійович",  
 "Юрійович", "Віталійович", "Миколайович",  
 "Євгенович", "Сергійович", "Олегович",  
 "Ілліч", "Іванович", "Костянтинович",  
 "Станіславович", "Андрійович",  
 "Володимирович", "Дмитрович",  
 "Вікторович"  
 )  
  
 name = f'{random.choice(last\_names)} ' \  
 f'{random.choice(first\_names)} ' \  
 f'{random.choice(patronymics)}'  
  
 *if* birthday *is None*:  
 random\_date\_start = datetime.date(year=1950, month=1, day=1)  
 random\_date\_end = datetime.date(2004, 12, 31)  
 random\_day = random.randint(  
 0,  
 (random\_date\_end - random\_date\_start).days  
 )  
 birthday = random\_date\_start + datetime.timedelta(days=random\_day)  
  
 *self*.\_\_name = name  
 *self*.\_\_birthday = birthday  
  
 @property  
 *def* name(*self*) -> str:  
 *return self*.\_\_name  
  
 @name.setter  
 *def* name(*self*, name: str):  
 *self*.\_\_name = name  
  
 @property  
 *def* birthday(*self*) -> datetime.date:  
 *return self*.\_\_birthday  
  
 @birthday.setter  
 *def* birthday(*self*, new\_date: datetime.date):  
 *self*.\_\_birthday = new\_date  
  
 *def* get\_age(*self*):  
 *return* datetime.datetime.fromtimestamp(  
 (datetime.date.today() - *self*.\_\_birthday).total\_seconds()  
 ).year - 1970  
  
 @abstractmethod  
 *def* get\_salary(*self*):  
 ...

Файл Student/\_\_init\_\_.py

*from* Person *import* Person  
*from* Scholarship *import* Scholarship  
*import* random  
  
*class* Student(Person):  
 *def* \_\_init\_\_(  
 *self*,  
 name=*None*,  
 birthday=*None*,  
 group\_name=*None*,  
 grade\_point\_average=*None* ):  
 super().\_\_init\_\_(name, birthday)  
  
 *if* group\_name *is None*:  
 group\_prefixes = (  
 "ІА", "ІК", "ІС", "ІП", "ІТ", "ІО", "ІМ", "ІВ"  
 )  
  
 group\_name = f'{random.choice(group\_prefixes)}-' \  
 f'{random.randint(11, 15)}'  
  
 *if* grade\_point\_average *is None*:  
 grade\_point\_average = round(random.uniform(60, 100), 2)  
  
 *self*.\_\_group\_name = group\_name  
 *self*.\_\_grade\_point\_average = grade\_point\_average  
  
 *def* get\_scholarship(*self*):  
 scholarship = Scholarship.NONE  
  
 *if* 95 <= *self*.\_\_grade\_point\_average:  
 scholarship = Scholarship.INCREASED  
 *elif* 85 <= *self*.\_\_grade\_point\_average:  
 scholarship = Scholarship.DEFAULT  
  
 *return* scholarship  
  
 *def* get\_salary(*self*):  
 scholarship = *self*.get\_scholarship()  
 salary = 0  
  
 *if* scholarship == Scholarship.DEFAULT:  
 salary = 2000  
 *elif* scholarship == Scholarship.INCREASED:  
 salary = 2550  
  
 *return* salary  
  
 @property  
 *def* group\_name(*self*):  
 *return self*.\_\_group\_name  
  
 @property  
 *def* get\_grade\_point\_average(*self*):  
 *return self*.\_\_grade\_point\_average  
  
 *def* \_\_str\_\_(*self*):  
 *return* f'ПІБ студента {*self*.name}\n' \  
 f'Дата народження: {*self*.birthday:%d.%m.%Y} ' \  
 f'(вік: {*self*.get\_age()})\n' \  
 f'Номер академічної групи: {*self*.\_\_group\_name}\n' \  
 f'Середній рейтинговий бал: {*self*.\_\_grade\_point\_average}\n' \  
 f'Стипендія: {*self*.get\_scholarship().value}\n' \  
 f'Місячний заробіток: {*self*.get\_salary()} грн.' \  
  
 *def* \_\_repr\_\_(*self*):  
 *return self*.\_\_str\_\_()

Файл Scholarship/\_\_init\_\_.py

*from* enum *import* Enum  
  
*class* Scholarship(Enum):  
 INCREASED = 'Підвищена'  
 DEFAULT = 'Звичайна'  
 NONE = 'Відсутня'

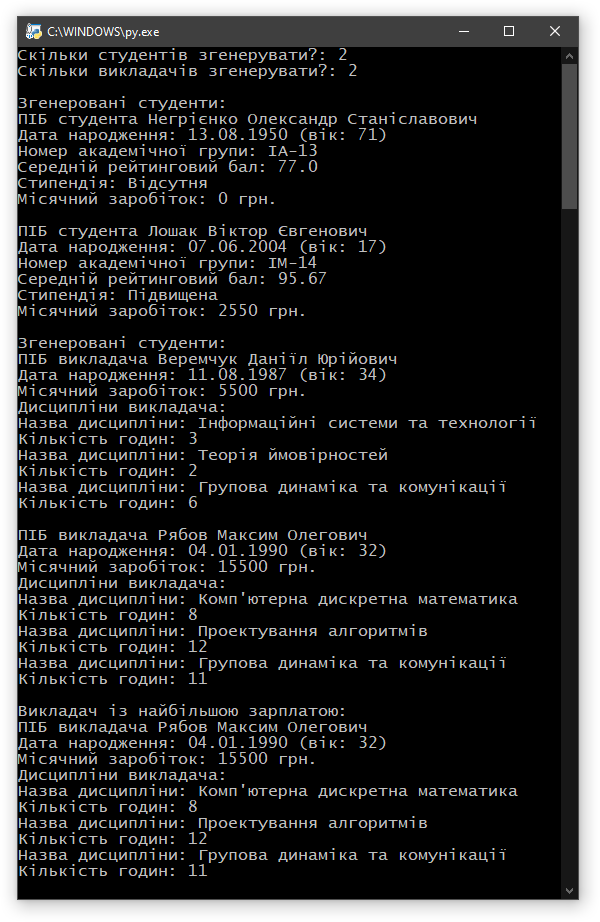
Файл Teacher/\_\_init\_\_.py

*import* random  
*import* typing  
  
*from* Person *import* Person  
*from* Discipline *import* Discipline  
  
*class* Teacher(Person):  
 *def* \_\_init\_\_(  
 *self*,  
 name=*None*,  
 birthday=*None*,  
 disciplines=*None* ):  
 super().\_\_init\_\_(name, birthday)  
  
 *if* disciplines *is None*:  
 disciplines = []  
 disciplines\_count = random.randint(1, 3)  
  
 *while* len(disciplines) != disciplines\_count:  
 discipline = Discipline()  
 *if not* discipline.name *in* [dsc.name *for* dsc *in* disciplines]:  
 disciplines.append(discipline)  
  
 *self*.\_\_disciplines: typing.List[Discipline] = disciplines  
  
  
 *def* get\_salary(*self*):  
 salary\_base = 500  
 total\_salary = 0  
  
 *for* discipline *in self*.\_\_disciplines:  
 total\_salary += discipline.number\_of\_hours \* salary\_base  
  
 *return* total\_salary  
  
  
 *def* add\_discipline(*self*, discipline: Discipline):  
 *self*.\_\_disciplines.append(discipline)  
 *return self  
  
  
 def* \_\_str\_\_(*self*):  
 result = f'ПІБ викладача {*self*.name}\n' \  
 f'Дата народження: {*self*.birthday:%d.%m.%Y} ' \  
 f'(вік: {*self*.get\_age()})\n' \  
 f'Місячний заробіток: {*self*.get\_salary()} грн.\n' \  
 f'Дисципліни викладача:\n'  
  
 *for* discipline *in self*.\_\_disciplines:  
 result += f'{discipline}\n'  
  
 *return* result  
  
  
 *def* \_\_repr\_\_(*self*):  
 *return self*.\_\_str\_\_()

Файл Discipline/\_\_init\_\_.py

*import* random  
  
*class* Discipline:  
 *def* \_\_init\_\_(  
 *self*,  
 name=*None*,  
 number\_of\_hours=*None* ):  
 *if* name *is None*:  
 discipline\_names = (  
 "Комп'ютерна дискретна математика",  
 "Математичний аналіз",  
 "Лінійна алгебра та аналітична геометрія",  
 "Українська мова за професійним спрямуванням",  
 "Екологічна безпека та цивільний захист",  
 "Історія науки і техніки",  
 "Основи здорового способу життя",  
 "Іноземна мова",  
 "Теорія ймовірностей",  
 "Групова динаміка та комунікації",  
 "Філософські основи наукового пізнання",  
 "Права і свободи людини",  
 "Економіка IT - індустрії та підприємництво",  
 "Іноземна мова професійного спрямування",  
 "Фізика",  
 "Алгоритми та структури даних",  
 "Основи програмування",  
 "Основи компютерних систем і мереж",  
 "Бази даних",  
 "Курсова робота з баз даних",  
 "Компоненти програмної інженерії",  
 "Безпека програмного забезпечення",  
 "Проектування алгоритмів",  
 "Системне програмне забезпечення",  
 "Програмування веб - застосувань",  
 "Ймовірністні моделі та статистичне оцінювання в інформаційних системах",  
 "Аналіз даних в інформаційних системах",  
 "Курсова робота з аналізу даних в інформаційних системах",  
 "Мультипарадигмене програмування",  
 "Програмування інтернет речей",  
 "Методи оптимізації та прийняття рішень",  
 "Технології паралельних обчислень",  
 "Програмування інтелектуальних інформаційних систем",  
 "Моделювання систем",  
 "Інформаційні системи та технології"  
 )  
 name = random.choice(discipline\_names)  
  
 *if* number\_of\_hours *is None*:  
 number\_of\_hours = random.randint(2, 12)  
  
 *self*.\_\_name = name  
 *self*.\_\_number\_of\_hours = number\_of\_hours  
  
  
 @property  
 *def* number\_of\_hours(*self*):  
 *return self*.\_\_number\_of\_hours  
  
  
 @property  
 *def* name(*self*):  
 *return self*.\_\_name  
  
  
 *def* \_\_str\_\_(*self*):  
 *return* f'Назва дисципліни: {*self*.\_\_name}\n' \  
 f'Кількість годин: {*self*.\_\_number\_of\_hours}'  
  
  
 *def* \_\_repr\_\_(*self*):  
 *return self*.\_\_str\_\_()

Виконання програми мовою Python:



Висновок: під час виконання даної лабораторної роботи я здобув практичних навичок із використання таких понять ООП, як поліморфізм та наслідування. Особливим методом у даній лабораторній роботі був метод int getSalary() в класі Person, адже він демонструє одразу обидва з цих принципів: поліморфізм, адже працює для різних типів даних по-різному, а наслідування, адже є чисто віртуальним методом, який можна використовувати лише при наслідуванні та який треба перевантажувати в кожному дочірньому класі, якщо він не повинен бути абстрактним, адже у даній лабораторній роботі навіть неможливо було створити об’єкт класу Person через чисто віртуальний метод