Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення

**Звіт**

Про виконання лабораторної роботи №9

**На тему:**

«Принцип поліморфізму»

з дисципліни

«Об’єктно-орієнтоване програмування»

**Лектор:**

Доцент каф. ПЗ

Коротєєва Т. О.

**Виконав:**

ст. гр. ПЗ-11

Солтисюк Д. А.

**Прийняла:**

Доцент каф. ПЗ

Коротєєва Т. О.

« \_\_ » \_\_\_\_\_\_\_\_ 2022 р.

∑ = \_\_\_\_\_ \_\_\_\_\_\_\_\_\_ .

Львів – 2022

**Тема**: Наслідування. Створення та використання ієрархії класів

**Мета**: Навчитися створювати базові та похідні класи, використовувати

наслідування різного типу доступу, опанувати принципи використання

множинного наслідування. Навчитися перевизначат и методи в похідному

класі, освоїти принципи такого перевизначення.

**Теоретичні відомості**

Наслідуванням називається процес визначення класу на основі іншого класу. На новий (дочірній) клас за замовчуванням поширюються всі визначення змінних екземпляра і методів зі старого (батьківського) класу, але можуть бути також визначені нові компоненти або «перевизначені» визначення батьківських функцій і дано нові визначення. Прийнято вважати, що клас А успадковує свої визначення від класу В, якщо клас А Визначений на основі класу В зазначеним способом. Класи можуть бути пов'язані один з одним різними відношеннями. При наслідуванні всі атрибути і методи батьківського класу успадковуються Класом нащадком. Наслідування може бути багаторівневим, і тоді класи, що знаходяться на нижніх рівнях ієрархії, успадкують всі властивості (атрибути і методи) всіх класів, прямими або непрямими нащадками яких вони є.

Крім одиничного, існує і множинне наслідування, коли клас наслідує відразу кілька класів. При цьому він успадкує властивості всіх класів, нащадком яких він є. Така зміна семантики методу називається поліморфізмом. Поліморфізм –це виконання методом з одним і тим же ім'ям різних дій залежно від контексту, зокрема, від приналежності до того чи іншого класу.

У різних мовах програмування поліморфізм реалізується різними способами.

Щоб привести вказівник базового типу до похідного типу використовуємо функцію dynamic\_cast. Якщо вказівник не вдалось привести до похідного типу функція верне нуль.

**Завдання. Варіант №9**

КлієнтГуртівні

Базовий клас – WholeSaleClient. Далі – RegularWSClient, VIPWSClient. Базовий клас зберігає

загальні дані про клієнтів. VIP до прикладу, дозволяє, швидше отримувати нотифікації, різні

варіанти накопичувальних знижок тощо.

**Хід роботи**

**Код програми:**

main.cpp:

#include <QApplication>

#include "widget.h"

int main(int argc, char \*argv[]) {

QApplication a(argc, argv);

Widget w;

w.show();

**return** a.exec();

}

wholesale-client.h:

#pragma once

#include <iostream>

#include <string>

**using** std::string;

*// Wholesale thing entity*

struct WholesaleThing {

string id;

string name;

double price;

};

template <typename BuyEntity> struct CanDoPurchases {

virtual void buy(BuyEntity thing) = 0;

};

*// Wholesale client general class, as base for other classes*

class WholesaleClient : CanDoPurchases<WholesaleThing> {

protected:

string name;

double balance;

double discount\_percent = 0;

bool notifiable = **false**;

public:

WholesaleClient(string name, double balance = 0) {

**this**->name = name;

**this**->balance = balance;

};

virtual string getClassName() { **return** "WholesaleClient"; };

double getPriceWithDiscount(double itemPrice) {

**return** itemPrice - (itemPrice \* **this**->discount\_percent);

}

void buy(WholesaleThing thing) override {

const auto priceWithDiscount = **this**->getPriceWithDiscount(thing.price);

**if** (**this**->balance < priceWithDiscount) {

**throw** std::invalid\_argument("Balance can't be lower than the price");

}

**this**->balance -= priceWithDiscount;

**if** (**this**->notifiable) {

std::cout << "The item with ID " << thing.id

<< " was successfully purchased";

}

**return**;

}

};

class RegularWholesaleClient : public WholesaleClient {

public:

RegularWholesaleClient(string name, double balance = 0)

: WholesaleClient(name, balance) {

**this**->discount\_percent = 0;

**this**->notifiable = **false**;

};

string getClassName() override { **return** "RegularWholesaleClient"; };

};

class ComplexWholesaleClient : public WholesaleClient {

public:

ComplexWholesaleClient(string name, double balance = 0)

: WholesaleClient(name, balance) {

**this**->discount\_percent = 0.05;

**this**->notifiable = **true**;

}

string getClassName() override { **return** "ComplexWholesaleClient"; }

};

class VipWholesaleClient : public ComplexWholesaleClient {

public:

VipWholesaleClient(string name, double balance = 0)

: ComplexWholesaleClient(name, balance) {

**this**->discount\_percent = 0.1;

**this**->notifiable = **true**;

}

string getClassName() override { **return** "VipWholesaleClient"; }

};

widget.cpp:

#include "widget.h"

#include "wholesale-client.h"

#include <QFile>

#include <QGridLayout>

#include <QTextStream>

#include <iostream>

#include <vector>

void Widget::on\_output() {

const string clientName = "Clementh";

const auto balanceForEveryone = 100;

const WholesaleThing thing = {"123", *// id*

"Shovel", *// name*

15};

**this**->operated\_classes.push\_back(

**new** RegularWholesaleClient(clientName, balanceForEveryone));

**this**->operated\_classes.push\_back(

**new** ComplexWholesaleClient(clientName, balanceForEveryone));

**this**->operated\_classes.push\_back(

**new** VipWholesaleClient(clientName, balanceForEveryone));

std::cout << (**new** ComplexWholesaleClient(clientName, balanceForEveryone))

->getClassName()

<< '\n'

<< **this**->operated\_classes[1]->getClassName();

**this**->class\_names\_output->setMarkdown(

QString("### WholesaleClient clild classes:\n\n"

"\* %1\n"

"\* %2\n"

"\* %3")

.arg(

QString::fromStdString(**this**->operated\_classes[0]->getClassName()))

.arg(

QString::fromStdString(**this**->operated\_classes[1]->getClassName()))

.arg(QString::fromStdString(

**this**->operated\_classes[2]->getClassName())));

auto output\_string =

QString("### Parameters\n"

"`balance = %4, wholesale thing price = %5`:\n"

"\n"

"### Result price\n"

"\* Regular: %1\n"

"\* Complex: %2\n"

"\* VIP: %3")

.arg(QString::number(

**this**->operated\_classes[0]->getPriceWithDiscount(thing.price)))

.arg(QString::number(

**this**->operated\_classes[1]->getPriceWithDiscount(thing.price)))

.arg(QString::number(

**this**->operated\_classes[2]->getPriceWithDiscount(thing.price)))

.arg(QString::number(balanceForEveryone))

.arg(QString::number(thing.price));

QFile file("results.md");

**if** (file.open(QIODevice::Append)) {

QTextStream stream(&file);

stream << output\_string << Qt::endl << Qt::endl;

file.close();

}

**this**->results\_output->setMarkdown(output\_string);

}

Widget::Widget(QWidget \*parent) : QWidget(parent) {

auto \*main\_layout = **new** QGridLayout;

**this**->output\_btn = **new** QPushButton("Print output");

**this**->class\_names\_output = **new** QTextEdit;

**this**->class\_names\_output->setReadOnly(**true**);

**this**->results\_output = **new** QTextEdit;

**this**->results\_output->setReadOnly(**true**);

main\_layout->addWidget(**this**->class\_names\_output, 0, 0);

main\_layout->addWidget(**this**->results\_output, 0, 1);

main\_layout->addWidget(**this**->output\_btn, 1, 0, 1, 2);

connect(**this**->output\_btn, &QPushButton::released, **this**, &Widget::on\_output);

setLayout(main\_layout);

}

widget.h:

#pragma once

#include "wholesale-client.h"

#include <QPushButton>

#include <QTextEdit>

#include <QWidget>

class Widget : public QWidget {

Q\_OBJECT

public:

Widget(QWidget \*parent = nullptr);

private slots:

void on\_output();

private:

std::vector<WholesaleClient \*> operated\_classes;

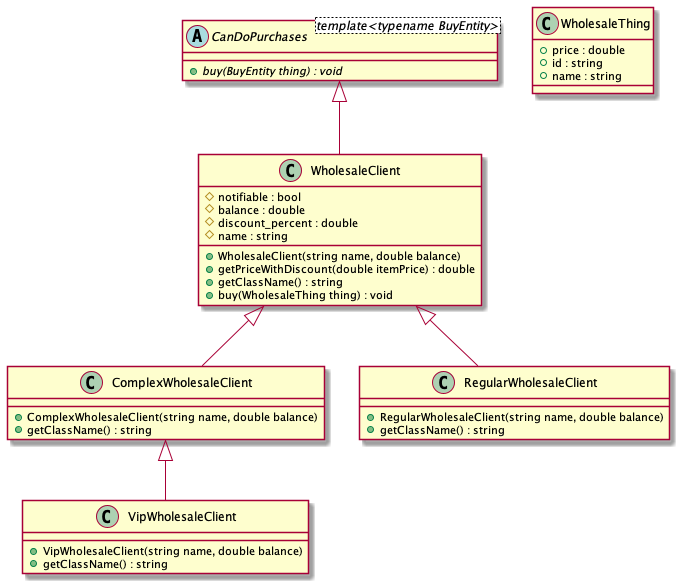
QPushButton \*output\_btn;

QTextEdit \*class\_names\_output;

QTextEdit \*results\_output;

};

**Діаграмма:**



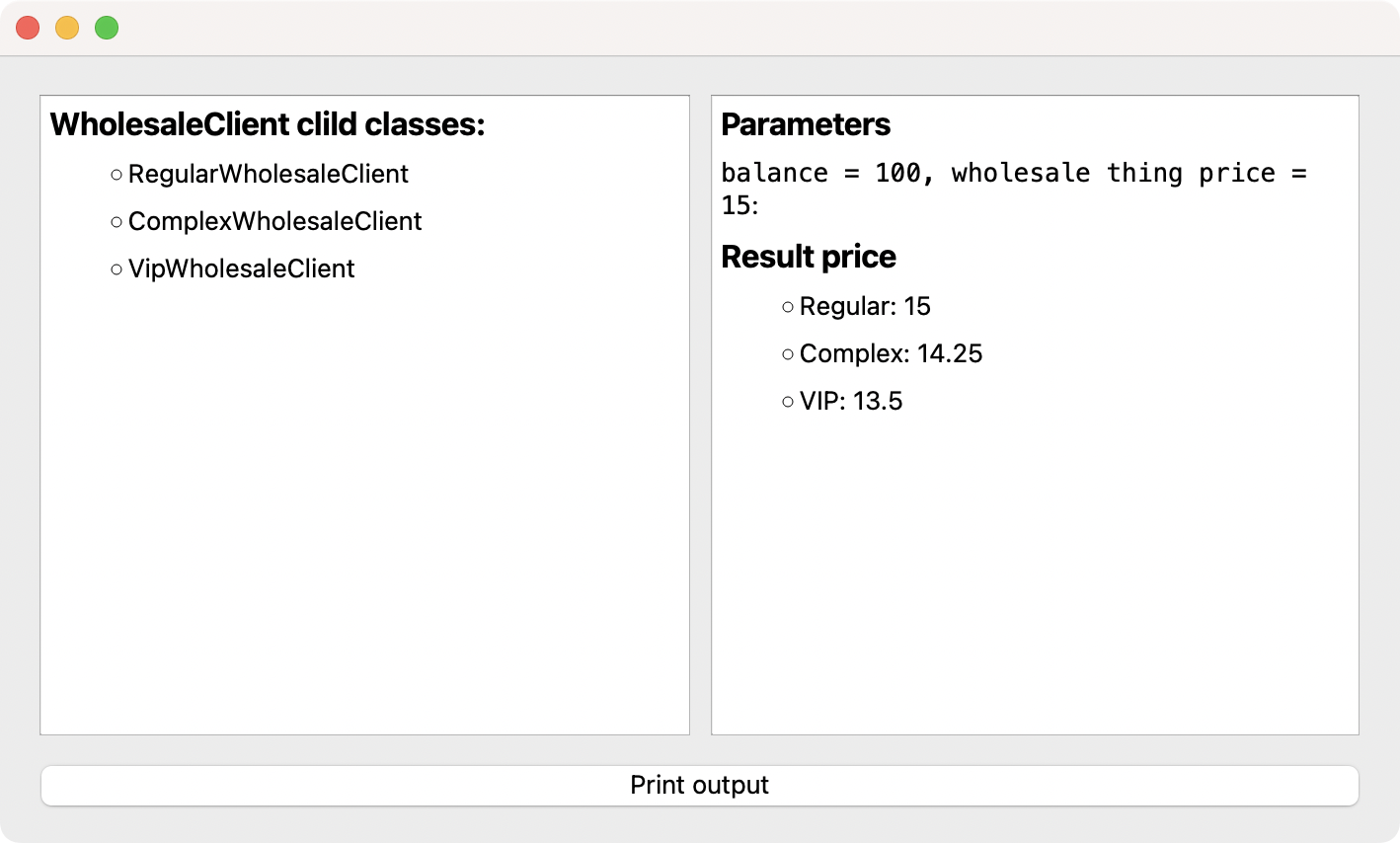
**Результати виконання програми**

Рис. 1. Результати обчислень програми

**Висновок**

Виконуючи цю лабораторну роботу, я навчився створювати базові та похідні класи,

Використовувати наслідування різного типу доступу, опанував принципи

використання множинного наслідування. Навчився

перевизначати методи в похідному класі, освоїв принципи такого перевизначення.