

Міністерство освіти і науки України  
Національний університет "Львівська політехніка"  
Інститут комп'ютерних наук та інформаційних технологій  
Кафедра програмного забезпечення

## **Звіт**

Про виконання лабораторної роботи №8

### **На тему:**

«Наслідування. Створення та використання ієрархії класів»  
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.

$\Sigma$  = \_\_\_\_\_ .

**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 перевизначати 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 непрямыми 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.

## Завдання. Варіант №3

3. Розробити ієрархію класів для сутності: **банківський депозит**.

Розробити такі типи депозитів:

- Строковий (виплата відсотків відбувається після закінчення терміну депозиту);
- Накопичувальний (капіталізація відсотків, виплата відбувається кожного місяця);
- VIP (капіталізація відсотків, виплата кожного місяця, можливість поповнення рахунку в будь-який день, збільшення відсоткової ставки із заданим коефіцієнтом при збільшенні суми вкладу (обмежене зверху)).

Всі класи повинні вміти обчислювати прибуток за вказаний та за весь період вкладу.

# Хід роботи

## Код програми:

bankdeposit.cpp:

```
#include "bankdeposit.h"

BankDeposit::BankDeposit(long _open_date, long _terminate_date, double _rate,
                        double _amount) {
    open_date = _open_date;
    termination_date = _terminate_date;
    rate = _rate;
    amount = _amount;
}

std::string BankDeposit::print_class_name() { return "BankDeposit"; }
```

bankdeposit.h:

```
#pragma once
#include <string>

class BankDeposit {
protected:
    long open_date;
    long termination_date;
    double rate;
    double amount;

public:
    BankDeposit(long _open_date, long _terminate_date, double _rate,
                double _amount);
    virtual double calculate_income(long to) = 0;
    virtual double calculate_income() = 0;

    virtual std::string print_class_name() = 0;
};
```

complexbankdeposit.cpp:

```
#include "complexbankdeposit.h"

ComplexBankDeposit::ComplexBankDeposit(long _open_date, long _terminate_date,
                                        double _rate, double _amount)
    : BankDeposit(_open_date, _terminate_date, _rate, _amount) {}

double ComplexBankDeposit::calculate_income(long to) {
    auto month_count = (to - this->open_date) / (60. * 60 * 24 * 30);
    auto progressive_amount = this->amount;
    for (auto i = 0; i < month_count; i++) {
        progressive_amount += progressive_amount * rate;
    }
    return progressive_amount - this->amount;
}

double ComplexBankDeposit::calculate_income() {
    return this->calculate_income(this->termination_date);
}
```

```

}

std::string ComplexBankDeposit::print_class_name() {
    return "ComplexBankDeposit";
}

```

complexbankdeposit.h:

```

#pragma once
#include "bankdeposit.h"

class ComplexBankDeposit : public BankDeposit {
public:
    ComplexBankDeposit(long _open_date, long _terminate_date, double _rate,
                       double _amount);
    std::string print_class_name() override;
    double calculate_income(long to) override;
    double calculate_income() override;
};

```

main.cpp:

```

#include <QApplication>

#include "widget.h"

int main(int argc, char *argv[]) {
    QApplication a(argc, argv);
    Widget w;
    w.show();
    return a.exec();
}

```

simplebankdeposit.cpp:

```

#include "simplebankdeposit.h"

SimpleBankDeposit::SimpleBankDeposit(long _open_date, long _terminate_date,
                                       double _rate, double _amount)
    : BankDeposit(_open_date, _terminate_date, _rate, _amount) {}

double SimpleBankDeposit::calculate_income(long to) {
    auto month_count = (to - this->open_date) / (60. * 60 * 24 * 30);
    return this->amount * rate * month_count;
}

double SimpleBankDeposit::calculate_income() {
    return this->calculate_income(this->termination_date);
}

std::string SimpleBankDeposit::print_class_name() {
    return "SimpleBankDeposit";
}

```

simplebankdeposit.h:

```

#pragma once
#include "bankdeposit.h"

```

```

class SimpleBankDeposit : public BankDeposit {
public:
    SimpleBankDeposit(long _open_date, long _terminate_date, double _rate,
                      double _amount);
    std::string print_class_name() override;
    double calculate_income(long to) override;
    double calculate_income() override;
};

```

vipbankdeposit.cpp:

```

#include "vipbankdeposit.h"

VIPBankDeposit::VIPBankDeposit(long _open_date, long _terminate_date,
                                double _rate, double _amount)
    : ComplexBankDeposit(_open_date, _terminate_date, _rate, _amount) {}

void VIPBankDeposit::add_money(double money) {
    this->amount += money;
    this->rate = std::min(.02, this->rate + money / 1000000);
}

std::string VIPBankDeposit::print_class_name() { return "VIPBankDeposit"; }

```

vipbankdeposit.h:

```

#pragma once
#include "complexbankdeposit.h"

class VIPBankDeposit : public ComplexBankDeposit {
public:
    VIPBankDeposit(long _open_date, long _terminate_date, double _rate,
                  double _amount);
    void add_money(double);
    std::string print_class_name() override;
};

```

widget.cpp:

```

#include "widget.h"

#include <QFile>
#include <QGridLayout>
#include <QTextStream>
#include <vector>

#include "bankdeposit.h"
#include "complexbankdeposit.h"
#include "simplebankdeposit.h"
#include "vipbankdeposit.h"

void Widget::on_output() {
    auto terminate_date = 60 * 60 * 24 * 30 * 3;
    auto monthly_rate = .01;
    auto initial_amount = 10000;

    SimpleBankDeposit d1(0, terminate_date, monthly_rate, initial_amount);
    ComplexBankDeposit d2(0, terminate_date, monthly_rate, initial_amount);
    VIPBankDeposit d3(0, terminate_date, monthly_rate, initial_amount);
}

```

```

d3.add_money(1000000);

this->class_names_output->setMarkdown(
    QString("### BankDeposit classes:\n\n"
        "* %1\n"
        "* %2\n"
        "* %3")
        .arg(QString::fromStdString(d1.print_class_name()))
        .arg(QString::fromStdString(d2.print_class_name()))
        .arg(QString::fromStdString(d3.print_class_name())));

auto output_string =
    QString("### Parameters\n"
        "`monthly rate = %4, initial amount = %5, months = %6`: \n"
        "\n"
        "### Results\n"
        "* Simple Deposit income: %1\n"
        "* Complex Deposit income: %2\n"
        "* VIP Deposit income: %3")
        .arg(QString::number(d1.calculate_income()))
        .arg(QString::number(d2.calculate_income()))
        .arg(QString::number(d3.calculate_income()))
        .arg(QString::number(monthly_rate))
        .arg(QString::number(initial_amount))
        .arg(QString::number(terminate_date / (60 * 60 * 24 * 30)));

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 and save to file");

    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 <QPushButton>
#include <QTextEdit>
#include <QWidget>

class Widget : public QWidget {
    Q_OBJECT

public:
    Widget(QWidget *parent = nullptr);

private slots:
    void on_output();

private:
    QPushButton *output_btn;

    QTextEdit *class_names_output;
    QTextEdit *results_output;
};
```

## Результати виконання програми

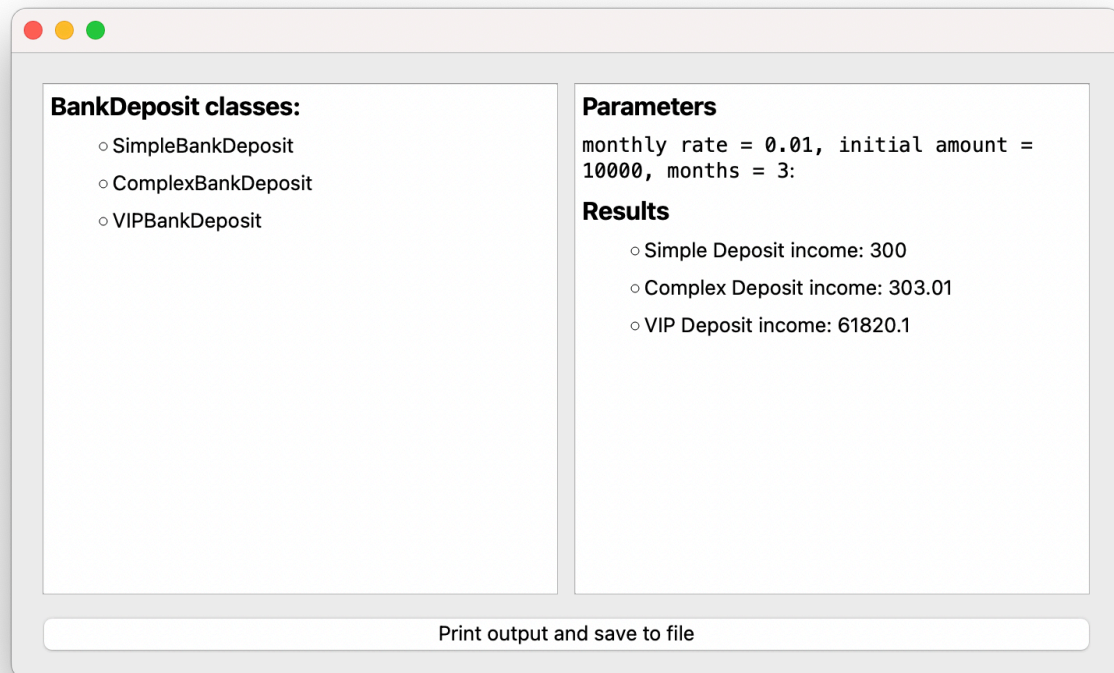


Рис. 1. Результати обчислень програми



```
### Parameters

monthly rate = 0.01, initial amount = 10000, months = 3 :

### Results

• Simple Deposit income: 300
• Complex Deposit income: 303.01
• VIP Deposit income: 61820.1
```

Рис. 2. Вивід результату у форматі markdown у файл

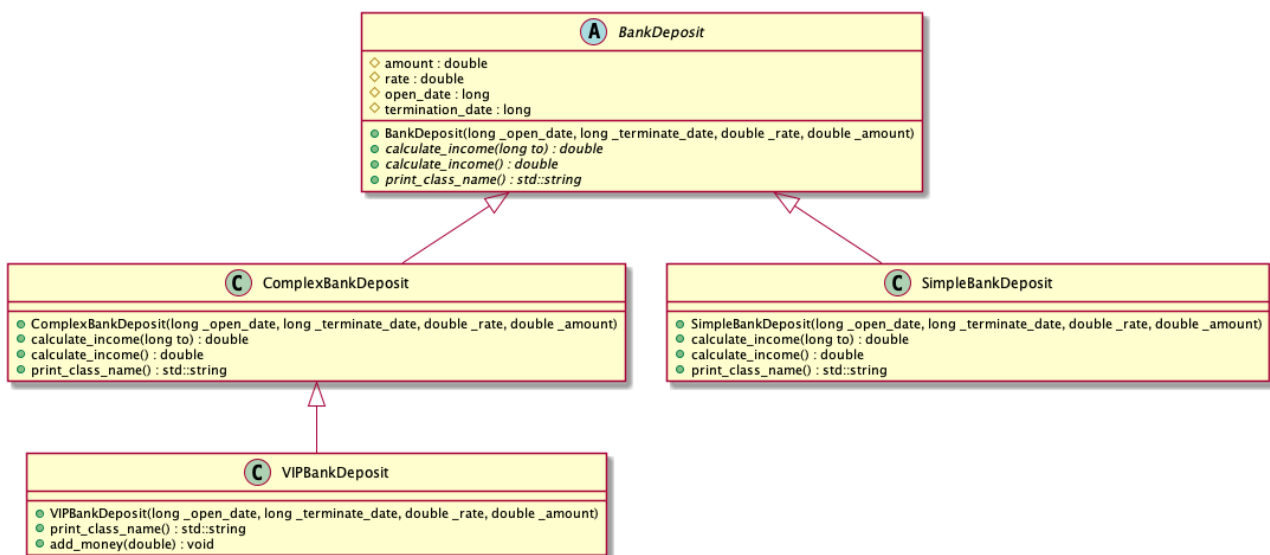


Рис. 3. Схема класів

## 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.