МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **КНІТ**
Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи № 8
На тему: "*Наслідування. Створення та використання ієрархії класів*"
З дисципліни: "Об'єктно-орієнтоване програмування"

Лектор:
доцент кафедри ПЗ
Коротєєва Т.О.

Виконав:
студент групи ПЗ-16
Коваленко Д.М.

Прийняв:
доцент кафедри ПЗ
Яцишин С.І.

«_____» _____ 2022 р.
$\sum$ = _____.....

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 перевизначати методи в похідному класі, освоїти принципи такого перевизначення.

Лабораторне завдання

1. Розробити ієрархію класів відповідно до варіанту
2. Створити базовий, похідні класи.
3. Використати public, protected наслідування
4. Використати множинне наслідування (за необхідності).
5. Виконати перевантаження функції print() в базовому класі, яка друкує назву відповідного класу, перевизначити її в похідних. В проекті при натисканні кнопки виведіть на форму назви всіх розроблених класів
6. Реалізувати методи варіанта та результати вивести на форму і у файл. При записі у файл використати різні варіанти аргументів конструктора
7. Оформити звіт до лабораторної роботи. Включити у звіт Uml-діаграму розробленої ієрархії класів.

Індивідуальне завдання

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.

Набір полів і методів, необхідних для забезпечення функціональної зручності класів, визначити самостійно.

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 A успадковує свої визначення від класу B, якщо клас A визначений на основі класу B зазначеним способом.

Класи можуть бути пов'язані один з одним різними відношеннями. Одним з основних є відношення клас-підклас, відоме в об'єктно-орієнтованому програмуванні як наслідування. Наприклад, клас автомобілів Audi 6 є підкласом легкових автомобілів, який в свою чергу входить у більший клас автомобілів, а останній є підкласом класу транспортних засобів, який крім автомобілів включає в себе літаки, кораблі, потяги і т.д. Прикладом подібних відношень є системи класифікації в ботаніці та зоології.

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 (рис. 1). При цьому він успадкує властивості всіх класів, нащадком яких він є.

При наслідуванні одні методи класу можуть заміщатися іншими. Так, клас транспортних засобів буде мати узагальнений метод руху. У класах-нащадках цей метод буде конкретизований: автомобіль буде їздити, літак – літати, корабель – плавати. 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.

Код програми

Назва файлу: *main.cpp*

```
#include "mainwindow.h"

#include <QApplication>

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

Назва файлу: *mainwindow.cpp*

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

#include "deposit.h"
#include "termdeposit.h"
#include "accumulativedeposit.h"
#include "vipdeposit.h"

#include "QIntValidator"
#include "QDoubleValidator"
#include "QMessageBox"
#include "QFileDialog"

MainWindow::MainWindow(QWidget *parent)
: QMainWindow(parent)
, ui(new Ui::MainWindow) {
    ui->setupUi(this);
}

MainWindow::~MainWindow() {
    delete ui;
}

Deposit *d;

void MainWindow::on_startButton_clicked() {
    if (
    ui->startMoneyEdit->text() == "" || ui->startTermEdit->text() == "" ||
    ui->startRateEdit->text() == "" ||
    (ui->typeComboBox->currentIndex() == 2 && (ui->startRateIncCoefEdit->text()
   == ""
    || ui->startMaxRateEdit->text() == ""))
    ) {
        QMessageBox msgBox;
        msgBox.setIcon(QMessageBox::Information);
        msgBox.setWindowTitle("Information");
        msgBox.setText("Fill in all the fields to get started!");
        msgBox.exec();
        return;
    }

    double money = ui->startMoneyEdit->text().toDouble();
    double term = ui->startTermEdit->text().toDouble();
    double rate = ui->startRateEdit->text().toDouble();
```

```
    if (ui->typeComboBox->currentIndex() == 0) {
        d = new TermDeposit(money, rate, term);
    } else if (ui->typeComboBox->currentIndex() == 1) {
        d = new AccumulativeDeposit(money, rate, term);
    } else if (ui->typeComboBox->currentIndex() == 2) {
        double rateIncCoef = ui->startRateIncCoefEdit->text().toDouble();
        double maxRate = ui->startMaxRateEdit->text().toDouble();
        d = new VIPDeposit(rateIncCoef, maxRate, money, rate, term);
    }
    ui->startGroupBox->setEnabled(false);
    ui->currentGroupBox->setEnabled(true);
    ui->moneyEdit->setText(QString::number(d->getMoney()));
    ui->rateEdit->setText(QString::number(d->getRate()));
}

void MainWindow::on_typeComboBox_currentIndexChanged(int index) {
    if (index == 2) {
        ui->startGroupBox->setFixedHeight(240);
        ui->currentGroupBox->setFixedHeight(180);
    } else if (index == 1) {
        ui->startGroupBox->setFixedHeight(180);
        ui->currentGroupBox->setFixedHeight(150);
    } else {
        ui->startGroupBox->setFixedHeight(180);
        ui->currentGroupBox->setFixedHeight(120);
    }
}

void MainWindow::on_addMoneyButton_clicked() {
    double money = ui->addMoneyEdit->text().toDouble();
    static_cast<VIPDeposit*>(d)->putMoney(money);
    static_cast<VIPDeposit*>(d)->setRate(d->getRate() + (static_cast<VIPDeposit
   *>(d)->getRateIncCoef() / 100) * money);
    if (d->getRate() > static_cast<VIPDeposit*>(d)->getMaxRate())
    static_cast<VIPDeposit*>(d)->setRate(static_cast<VIPDeposit*>(d)->getMaxRate
    ());
    ui->moneyEdit->setText(QString::number(d->getMoney()));
    ui->rateEdit->setText(QString::number(d->getRate()));
}

void MainWindow::on_profitButton_clicked() {
    if (ui->typeComboBox->currentIndex() == 0) {
        if (ui->profitMonthsEdit->text() == "") {
            ui->profitEdit->setText(QString::number(static_cast<TermDeposit*>(d)
   ->calculateProfit()));
            ui->paidEdit->setText(QString::number(static_cast<TermDeposit*>(d)->
    calculateProfit()));
        }
        else {
            ui->profitEdit->setText(QString::number(static_cast<TermDeposit*>(d)
   ->calculateProfit(ui->profitMonthsEdit->text().toDouble())));
            ui->paidEdit->setText(QString::number(static_cast<TermDeposit*>(d)->
    calculateProfit(ui->profitMonthsEdit->text().toDouble())));
        }
    } else if (ui->typeComboBox->currentIndex() == 1) {
        if (ui->profitMonthsEdit->text() == "") {
            ui->profitEdit->setText(QString::number(static_cast<
    AccumulativeDeposit*>(d)->calculateProfit()));
            ui->paidEdit->setText(QString::number(static_cast<
    AccumulativeDeposit*>(d)->calculateProfit()));
```

```
        }
        else {
            ui->profitEdit->setText(QString::number(static_cast<
    AccumulativeDeposit*>(d)->calculateProfit(ui->profitMonthsEdit->text().
    toDouble())));
            ui->paidEdit->setText(QString::number(static_cast<
    AccumulativeDeposit*>(d)->calculateProfit(ui->profitMonthsEdit->text().
    toDouble())));
        }
    } else if (ui->typeComboBox->currentIndex() == 2) {
        if (ui->profitMonthsEdit->text() == "") {
            QString profit = QString::number(static_cast<VIPDeposit*>(d)->
    calculateProfit());
            ui->profitEdit->setText(profit);
            ui->paidEdit->setText(profit);
        }
        else {
            ui->profitEdit->setText(QString::number(static_cast<VIPDeposit*>(d)
    ->calculateProfit(ui->profitMonthsEdit->text().toDouble())));
            ui->paidEdit->setText(QString::number(static_cast<VIPDeposit*>(d)->
    calculateProfit(ui->profitMonthsEdit->text().toDouble())));
        }
    }
}

void MainWindow::on_clearButton_clicked() {
    ui->startGroupBox->setEnabled(true);
    delete d;

    ui->startMaxRateEdit->clear();
    ui->startMoneyEdit->clear();
    ui->startRateEdit->clear();
    ui->startRateIncCoefEdit->clear();
    ui->startTermEdit->clear();

    ui->moneyEdit->clear();
    ui->paidEdit->clear();
    ui->profitEdit->clear();
    ui->profitMonthsEdit->clear();
    ui->rateEdit->clear();
    ui->addMoneyEdit->clear();

    ui->currentGroupBox->setEnabled(false);
}

void MainWindow::on_printButton_clicked() {
    ui->printLabel->setText(Deposit::print()+" "+TermDeposit::print()+" "+
    AccumulativeDeposit::print()+" "+VIPDeposit::print());
}

void MainWindow::on_writeButton_clicked() {
    QString filename = QFileDialog::getSaveFileName(this, "Save As");

    if (filename.isEmpty())
    return;

    QFile file(filename);
    if (!file.open(QIODevice::WriteOnly | QIODevice::Text))
    return;

    QTextStream out(&file);
```

```
    out << "Money:" << d->getMoney() << " Rate:" << d->getRate() << " Term:" <<
    d->getTerm() << "\n";
    file.close();
}
```

Назва файлу: *deposit.h*

```
#ifndef DEPOSIT_H
#define DEPOSIT_H

#include "QString"

class Deposit
{
    protected:
    double money;
    double rate;
    int term;
    public:
    Deposit();
    Deposit(double money, double rate, int term):
    money(money),
    rate(rate),
    term(term)
    {};
    double getMoney() { return money; };
    double getRate() { return rate; };
    int getTerm() { return term; };

    static QString print() { return "Deposit"; };
};

#endif // DEPOSIT_H
```

Назва файлу: *deposit.cpp*

```
#include "deposit.h"

Deposit::Deposit() {}
```

Назва файлу: *termdeposit.h*

```
#ifndef TERMDEPOSIT_H
#define TERMDEPOSIT_H

#include "deposit.h"

class TermDeposit : public Deposit
{
    public:
    TermDeposit();
    TermDeposit(double money, double rate, int term):
    Deposit(money, rate, term)
    {};

    static QString print() { return "TermDeposit"; };

    double calculateProfit();
    double calculateProfit(int months);
};

#endif // TERMDEPOSIT_H
```

Назва файлу: *termdeposit.cpp*

```cpp
#include "termdeposit.h"

TermDeposit::TermDeposit() {}

double TermDeposit::calculateProfit() {
    return money * (rate / 100);
}

double TermDeposit::calculateProfit(int months) {
    if (months <= term)
    return (money * (rate / 100)) * (months / double(term));
    else
    return money * (rate / 100);
}
```

Назва файлу: *accumulativedeposit.h*

```cpp
#ifndef ACCUMULATIVEDEPOSIT_H
#define ACCUMULATIVEDEPOSIT_H

#include "deposit.h"

class AccumulativeDeposit : public Deposit
{
    public:
    AccumulativeDeposit();
    AccumulativeDeposit(double money, double rate, int term):
    Deposit(money, rate, term)
    {};

    static QString print() { return "AccumulativeDeposit"; };

    double calculateProfit();
    double calculateProfit(int months);
};

#endif // ACCUMULATIVEDEPOSIT_H
```

Назва файлу: *accumulativedeposit.cpp*

```cpp
#include "accumulativedeposit.h"

AccumulativeDeposit::AccumulativeDeposit() {}

double AccumulativeDeposit::calculateProfit() {
    double money = this->money;
    for (int m = 0; m < term; m++) {
        money += (money * (rate / 100)) / 2;
    }
    return (money * (rate / 100)) / 2;
}

double AccumulativeDeposit::calculateProfit(int months) {
    double money = this->money;
    for (int m = 0; m < months; m++) {
        money += (money * (rate / 100)) / 2;
    }
    return (money * (rate / 100)) * (months / double(term)) / 2;
}
```

Назва файлу: *vipdeposit.h*

```
#ifndef VIPDEPOSIT_H
#define VIPDEPOSIT_H

#include "deposit.h"

class VIPDeposit : public Deposit {
    private:
    double rateIncCoef;
    double maxRate;
    public:
    VIPDeposit();
    VIPDeposit(double rateIncCoef, double maxRate, double money, double rate,
    int term):
    Deposit(money, rate, term),
    rateIncCoef(rateIncCoef),
    maxRate(maxRate)
    {};

    double getRateIncCoef() { return rateIncCoef; };
    double getMaxRate() { return maxRate; };


    void setMoney(double money) { this->money = money; };
    void setRate(double rate) { this->rate = rate; };
    void setRateIncCoef(double rateIncCoef) { this->rateIncCoef = rateIncCoef;
    };
    void setMaxRate(double maxRate) { this->maxRate = maxRate; };

    void putMoney(double money) { Deposit::money = Deposit::money + money; };
    static QString print() { return "VIPDeposit"; };

    double calculateProfit();
    double calculateProfit(int months);
};

#endif // VIPDEPOSIT_H
```

Назва файлу: *vipdeposit.cpp*

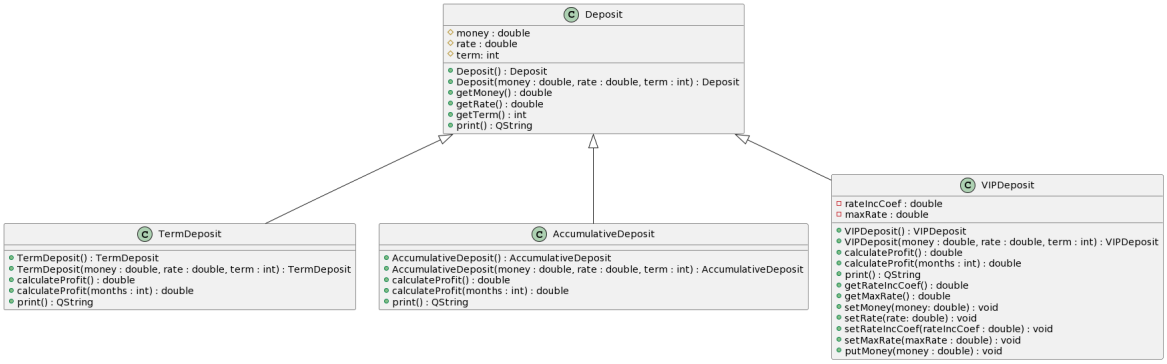
```
#include "vipdeposit.h"

VIPDeposit::VIPDeposit() {}

double VIPDeposit::calculateProfit() {
    double money = this->money;
    for (int m = 0; m < term; m++) {
        money += (money * (rate / 100)) / 2;
    }
    return (money * (rate / 100)) / 2;
}

double VIPDeposit::calculateProfit(int months) {
    double money = this->money;
    for (int m = 0; m < months; m++) {
        money += (money * (rate / 100)) / 2;
    }
    return (money * (rate / 100)) * (months / double(term)) / 2;
}
```

UML-діаграма



Висновок

Під час виконання лабораторної роботи я навчився створювати базові та похідні класи, використовувати наслідування різного типу доступу, опанувати принципи використання множинного наслідування. Навчився перевизначати методи в похідному класі, освоїв принципи такого перевизначення.