МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

ЗВІТ

До лабораторної роботи № 8

На тему: «Наслідування. Створення та використання ієрархії класів»

З дисципліни: «Об'єктно-орієнтоване програмування»

Лектор:
доц. кафедри ПЗ
Коротєєва Т. О.

Виконав:
ст. гр. ПЗ-16
Чаус О. М.

Прийняв:
асист. кафедри ПЗ
Дивак І. В.

« ____ » _____ 2022 р.

∑= _____ .....:

Львів-2022

Тема роботи: Наслідування. Створення та використання ієрархії класів.

Мета роботи: 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

Наслідування

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. 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.

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.

Private члени класу недоступні для наслідування. Звісно можна було б їх зробити public, та в такому випадку вони будуть доступні всім класам які знаходяться в програмі. Для того, щоб доступ до членів класу мали тільки класи нащадки потрібно використовувати модифікатор доступу protected.

Заміщення функцій

Об’єкт похідного класу має доступ до всіх функцій-членів базового класу а також до будь-якої функції члену яка оголошена в цьому ж класі. Крім цього базові функції можуть бути заміщені в похідному класі. Під заміщенням базової функції розуміють зміну її виконання в похідному класі.

Для заміщення необхідно описати функцію в похідному класі з таким же ж іменем як у базовому.

Недоліки заміщення.

Якщо в базовому класі у нас є перевантажена функція, яку ми хочемо замінити в похідному класі – ми не зможемо викликати в похідному класі будь-яку з цих перевантажених функцій.

Параметр рівня доступу при наслідуванні

Рівень доступу визначає статус членів базового класу в похідному класі. Як цей параметр використовуються специфікатори **public**, **private** або **protected**. Якщо рівень доступу не вказаний, то для похідного класу за умовчанням використовується специфікатор **private**, а для похідної структури - **public**.

Розглянемо варіанти, що виникають в цих ситуаціях. Якщо рівень доступу до членів базового класу задається специфікатором **public** то всі відкриті і захищені члени базового класу стають

відкритими і захищеними членами похідного класу. При цьому закриті члени базового класу не міняють свого статусу і залишаються недоступними членам похідного.
Якщо властивості базового класу успадковуються за допомогою специфікатора доступу **private**, всі відкриті і захищені члени базового класу стають закритими членами похідного класу. При закритому наслідуванні всі відкриті і захищені члени базового класу стають закритими членами похідного класу. Це означає, що вони залишаються доступними членам похідного класу, але недоступні решті елементів програми, що не є членами базового або похідного класів.
Специфікатор **protected** підвищує гнучкість механізму наслідування. Якщо член класу оголошений захищеним (protected), то поза класом він недоступний. З цієї точки зору захищений член класу нічим не відрізняється від закритого. Єдине виключення з цього правила стосується наслідування. У цій ситуації захищений член класу істотно відрізняється від закритого. Як вказувалося вище, закритий член базового класу не доступний іншим елементам програми, включаючи похідний клас. Проте захищені члени базового класу поводяться інакше. При відкритому наслідуванні захищені члени базового класу стають захищеними членами похідного класу до отже, доступні решті членів похідного класу. Іншими словами захищені члени класу по відношенню до свого класу є закритими і в той же час, можуть успадковуватися похідним класом.

Завдання для лабораторної роботи

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.

2. Розробити ієрархію класів для сутності: **банківський рахунок**.
Розробити наступні типи банківських рахунків:
- Звичайний (стандартна комісія на оплату комунальних послуг, перерахунок на інший рахунок, зняття готівки)
- Соціальний (оплата комунальних послуг безкоштовна, відсутня комісія за зняття готівки(пенсії), нараховується невеликий відсоток з залишку на картці)
- VIP (наявність кредитного ліміту, низький відсоток за користування кредитним лімітом, нараховуєтся більший відсоток, якщо залишок на картці більший за якусь суму)

Кожен із рахунків повинен зберігати історію транзакцій.
Набір полів і методів, необхідних для забезпечення функціональної зручності класів, визначити самостійно.

Хід роботи

Файл **BankAccount.h**:

```
#ifndef BANKACCOUNT_H
#define BANKACCOUNT_H
#include <QLabel>
#include <QTextEdit>
#include <QString>
```

```
class BankAccount
{
protected:
    ~BankAccount() = default;
    int monthCount;
    static QVector<QString> Log;
    QVector<double> localLog;
    double m_billsRate;
    double m_WithdrawRate;
    double m_balance;
    double m_bills;
    int m_id;
public:
    BankAccount(double, double);
    static void printLog(QTextEdit *);
    static bool printLog(const QString);
    bool Withdraw(double);
    bool PayBills();
    bool TransferTo(double, BankAccount *);
    void SkipMonth();
    virtual void print(QLabel * label) const = 0;
    void printLocalLog(QTextEdit *) const;
    bool printLocalLog(const QString) const;
    int getId() const;
    double getBalance() const;
    double getBills() const;
    void setBalance(double);
};

class CommonBankAccount : public BankAccount
{
public:
    CommonBankAccount();
    void print(QLabel *) const;
};

class SocialBankAccount : public BankAccount
{
private:
    double m_interest;
public:
    SocialBankAccount();
    void print(QLabel *) const;
    void SkipMonth();
};

class VIPBankAccount : public BankAccount
{
private:
    static double creditLimit;
    double m_creditBalance;
    double m_creditTaken;
    double m_currentCredit;
    double m_creditInterest;
public:
    VIPBankAccount();
    static double getCreditLimit();
    void print(QLabel *) const;
    bool takeCredit(double);
    void SkipMonth();
    double getCredit() const;
    double getCreditTaken() const;
    double getCreditBalance() const;
};

#endif // BANKACCOUNT_H
```

Файл **BankAccount.cpp**:

```
#include "BankAccount.h"
#include <QTime>
#include <string>
#include <random>
#include <QFile>

QVector<QString> BankAccount::Log;

double VIPBankAccount::creditLimit = 10000;

BankAccount::BankAccount(double bills, double withdraw) :
    m_billsRate(bills), m_WithdrawRate(withdraw), m_balance(0), m_bills(0)
{
    this->monthCount = 0;
    std::random_device r;
    std::default_random_engine en(r());
    std::uniform_int_distribution<int> uniform_dist(1000, 9999);
    this->m_id = uniform_dist(en);
}

void BankAccount::printLog(QTextEdit * textEdit)
{
    textEdit->clear();
    for(auto x : Log)
    {
        textEdit->append( x + "\n");
    }
}

bool BankAccount::printLog(const QString filePath)
{
    QFile file(filePath);
    if(!file.open(QFile::WriteOnly | QFile::Text))
        return false;
    QTextStream output(&file);
    for(auto x : Log)
    {
        output << ((x) + "\n");
    }
    return true;
}

void BankAccount::printLocalLog(QTextEdit * textEdit) const
{
    textEdit->clear();
    for(auto x : localLog)
    {
        QString sign = "";
        if(x > 0)
            sign = "+";
        textEdit->append("[MONTH" + QString::number(this->monthCount) + "]\t" + sign +
QString::number(x) + "\n");
    }
}

bool BankAccount::printLocalLog(const QString filePath) const
{
    {
        QFile file(filePath);
        if(!file.open(QFile::WriteOnly | QFile::Text))
            return false;
        QTextStream output(&file);
        for(auto x : localLog)
        {
            QString sign = "";
            if(x > 0)
```

```
                sign = "+";
            output << (sign + QString::number(x) + "\n");
        }
        return true;
    }

}

bool BankAccount::Withdraw(double amount)
{
    if(!(this->m_bills) && amount)
    {
        if(this->m_balance - (amount + amount * this->m_WithdrawRate) >= 0)
        {
            this->m_balance -= amount + amount * this->m_WithdrawRate;
            Log.push_back("[MONTH" + QString::number(this->monthCount) + "]\t" +
QString::number(this->m_id) + ": -" + QString::number(amount) + ", total: " +
QString::number(this->m_balance));
            localLog.push_back(-1 * amount);
            return true;
        }
    }
    return false;
}

bool BankAccount::PayBills()
{
    if(this->m_bills && this->m_balance)
    {
        if((this->m_balance - (this->m_bills + this->m_bills * this->m_billsRate)) >= 0)
        {
            this->m_balance -= this->m_bills + this->m_bills * this->m_billsRate;
            Log.push_back("[MONTH" + QString::number(this->monthCount) + "]\t" +
QString::number(this->m_id) + ": -" + QString::number(this->m_bills + this->m_bills * this-
>m_billsRate) +
                          ", total: " + QString::number(this->m_balance));
            localLog.push_back(-1 * (this->m_bills + this->m_bills * this->m_billsRate));
            this->m_bills = 0;
            return true;
        }
        else
        {
            this->m_bills -= this->m_balance;
            this->m_balance = 0;
            return true;
        }
    }
    return false;
}

bool BankAccount::TransferTo(double amount, BankAccount *receiver)
{
    if(!(this->m_bills) && amount)
    {
        if((this->m_balance - amount) > 0)
        {
            this->m_balance -= amount;
            Log.push_back("[MONTH" + QString::number(this->monthCount) + "]\t" +
QString::number(this->m_id) + ": -" + QString::number(amount) + ", total: " +
QString::number(this->m_balance));
            localLog.push_back(-1 * amount);
            receiver->setBalance(receiver->getBalance() + amount);
            Log.push_back("[MONTH" + QString::number(this->monthCount) + "]\t" +
QString::number(receiver->m_id) + ": +" + QString::number(amount) + ", total: " +
QString::number(receiver->m_balance));
            receiver->localLog.push_back(amount);
            return true;
        }
```

```
    }
    return false;
}

void BankAccount::SkipMonth()
{
    this->monthCount++;
    std::random_device r;
    std::default_random_engine en(r());
    std::uniform_int_distribution<int> bills(100, 300);
    std::uniform_int_distribution<int> salary(100, 500);
    this->m_bills += bills(en);
    double monthlySalary = salary(en);
    this->m_balance += monthlySalary;
    Log.push_back("[MONTH" + QString::number(monthCount) + "]\t" + QString::number(this-
>m_id) + ": +" + QString::number(monthlySalary) + ", total: " + QString::number(this-
>m_balance));
    localLog.push_back(monthlySalary);
}

int BankAccount::getId() const
{
    return this->m_id;
}

double  BankAccount::getBalance() const
{
    return this->m_balance;
}

double BankAccount::getBills() const
{
    return this->m_bills;
}
void BankAccount::setBalance(double amount)
{
    this->m_balance = amount;
}
CommonBankAccount::CommonBankAccount() : BankAccount::BankAccount(0.05, 0.1)
{
    std::string id = std::to_string(this->m_id);
    id[3] = '0';
    this->m_id = std::stoi(id);
}

void CommonBankAccount::print(QLabel * label) const
{
    label->setText("Common");
}

SocialBankAccount::SocialBankAccount() : BankAccount::BankAccount(0, 0), m_interest(0.01)
{
    std::string id = std::to_string(this->m_id);
    id[3] = '1';
    this->m_id = std::stoi(id);
}

void SocialBankAccount::print(QLabel * label) const
{
    label->setText("Social");
}

void SocialBankAccount::SkipMonth()
{
    this->monthCount++;
    std::random_device r;
    std::default_random_engine en(r());
    std::uniform_int_distribution<int> salary(100, 300);
```

```
    double monthlySalary = salary(en);
    this->m_balance += monthlySalary;
    Log.push_back("[MONTH" + QString::number(monthCount) + "]\t" + QString::number(this-
>m_id) + ": +" + QString::number(monthlySalary) + ", total: " + QString::number(this-
>m_balance));
    localLog.push_back(monthlySalary);
    double monthlyInterest = m_balance * m_interest;
    this->m_balance += m_balance * m_interest;
    Log.push_back("[MONTH" + QString::number(monthCount) + "]\t" + QString::number(this-
>m_id) + ": +" + QString::number(monthlyInterest) + ", total: " + QString::number(this-
>m_balance));
    localLog.push_back(monthlyInterest);
}

VIPBankAccount::VIPBankAccount() : BankAccount::BankAccount(0.05, 0.1), m_creditBalance(0),
m_creditInterest(0.2)
{
    this->m_creditTaken = 0;
    std::string id = std::to_string(this->m_id);
    id[3] = '2';
    this->m_id = std::stoi(id);
};

double VIPBankAccount::getCreditLimit()
{
    return creditLimit;
}

void VIPBankAccount::print(QLabel * label) const
{
    label->setText("VIP");
}

bool VIPBankAccount::takeCredit(double amount)
{
    if(!this->m_currentCredit && amount <= creditLimit)
    {
        this->m_creditTaken = amount;
        this->m_currentCredit = amount;
        if(m_balance > 2000)
            m_currentCredit *= 1.1;
        else m_currentCredit *= 1.05;
        this->m_balance += amount;
        Log.push_back("[MONTH" + QString::number(this->monthCount) + "]\t" +
QString::number(this->m_id) + ": +" + QString::number(amount) + ", total: " +
QString::number(this->m_balance));
        localLog.push_back(amount);
        this->m_creditBalance = m_currentCredit;
        return true;
    }
    return false;
}
void VIPBankAccount::SkipMonth()
{
    this->monthCount++;
    std::random_device r;
    std::default_random_engine en(r());
    std::uniform_int_distribution<int> bills(100, 300);
    std::uniform_int_distribution<int> salary(100, 1000);
    this->m_bills += bills(en);
    double monthlySalary = salary(en);
    this->m_balance += monthlySalary;
    Log.push_back("[MONTH" + QString::number(this->monthCount) + "]\t" +
QString::number(this->m_id) + ": +" + QString::number(monthlySalary) + ", total: " +
QString::number(this->m_balance));
    localLog.push_back(monthlySalary);
    if(this->m_currentCredit)
    {
```

```
        double payment = this->m_currentCredit * this->m_creditInterest;
        if(payment < 100)
            payment = 100;
        if(payment > this->m_creditBalance)
            payment = this->m_creditBalance;
        if(this->m_balance - payment >= 0)
        {
            if(this->m_creditBalance - payment >= 0)
            {
                this->m_balance -= payment;
                Log.push_back("[MONTH" + QString::number(this->monthCount) + "]\t" +
QString::number(this->m_id) + ": -" + QString::number(payment)
                              + ", total: " + QString::number(this->m_balance));
                localLog.push_back(payment);
                this->m_creditBalance -= payment;
            }
            else
            {
                this->m_balance -= m_creditBalance;
                Log.push_back("[MONTH" + QString::number(this->monthCount) + "]\t" +
QString::number(this->m_id) + ": -" + QString::number(this->m_creditBalance)
                              + ", total: " + QString::number(this->m_balance));
                this->m_creditBalance = 0;
            }
        }
        else
        {
            this->m_creditBalance -= this->m_balance;
            this->m_balance = 0;
            Log.push_back("[MONTH" + QString::number(this->monthCount) + "]\t" +
QString::number(this->m_id) + ": -" + QString::number(this->m_balance)
                          + ", total: " + "0");
        }
        if(this->m_creditBalance <= 0)
        {
            this->m_creditTaken = 0;
            this->m_creditBalance = 0;
            this->m_currentCredit = 0;
        }
    }
}

double VIPBankAccount::getCredit() const
{
    return m_currentCredit;
}
double VIPBankAccount::getCreditTaken() const
{
   return this->m_creditTaken;
}
double VIPBankAccount::getCreditBalance() const
{
    return m_creditBalance;
}
```

Файл **mainwindow.cpp**:

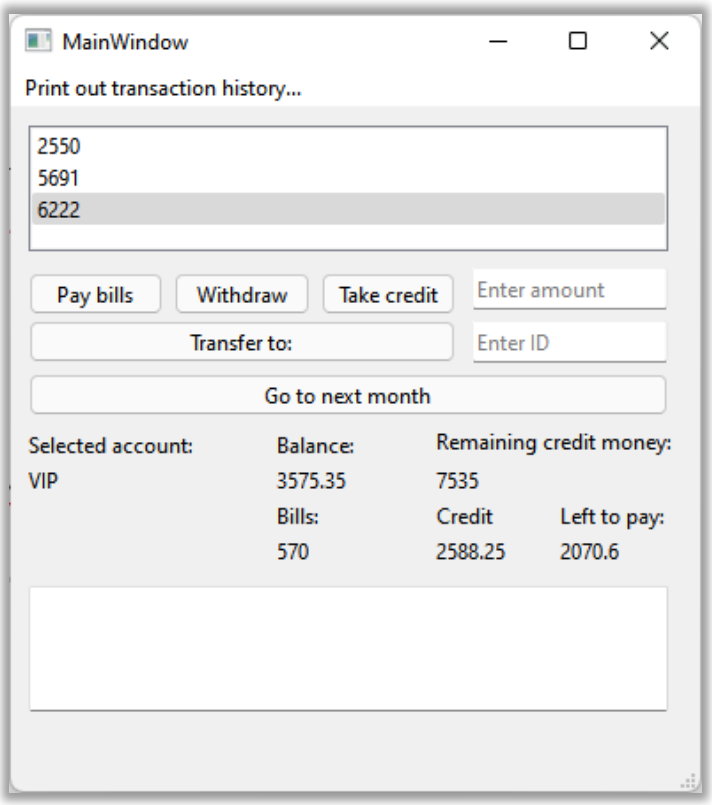
```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "BankAccount.h"
#include <QDoubleValidator>
#include <QIntValidator>
#include <QMessageBox>
#include <QLocale>
#include <QFileDialog>
#include <QDir>

CommonBankAccount common;
SocialBankAccount social;
```

```
VIPBankAccount vip;

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ui->lineID->setPlaceholderText("Enter ID");
    ui->lineMoney->setPlaceholderText("Enter amount");
    QDoubleValidator *validatorDouble = new QDoubleValidator(0, 10000, 2);
    validatorDouble->setLocale(QLocale::English);
    QIntValidator *validatorInt = new QIntValidator(1000, 9999);
    ui->lineID->setValidator(validatorInt);
    ui->lineMoney->setValidator(validatorDouble);
    QListWidgetItem *item = new QListWidgetItem();
    item->setText(QString::number(common.getId()));
    ui->listWidget->addItem(item);
    QListWidgetItem *item1 = new QListWidgetItem();
    item1->setText(QString::number(social.getId()));
    ui->listWidget->addItem(item1);
    QListWidgetItem *item2 = new QListWidgetItem();
    item2->setText(QString::number(vip.getId()));
    ui->listWidget->addItem(item2);
}

MainWindow::~MainWindow()
{
    delete ui;
}
void MainWindow::errorMessage(const QString str)
{
    QMessageBox *msg = new QMessageBox();
    msg->setText(str);
    msg->exec();
    delete msg;
}
void MainWindow::updateData()
{
    ui->lineMoney->clear();
    ui->lineID->clear();
    if(ui->listWidget->selectedItems().size())
    {
        if(ui->listWidget->currentItem()->text() == QString::number(common.getId()))
        {
            common.print(ui->labelName);
            ui->labelBalance->setText(QString::number(common.getBalance()));
            ui->labelBills->setText(QString::number(common.getBills()));
            ui->labelCredit->setText("");
            ui->labelRemainingCr->setText("");
            ui->bTakeCredit->setEnabled(false);
        }
        else if (ui->listWidget->currentItem()->text() == QString::number(social.getId()))
        {
            social.print(ui->labelName);
            ui->labelBalance->setText(QString::number(social.getBalance()));
            ui->labelBills->setText(QString::number(social.getBills()));
            ui->labelCredit->setText("");
            ui->labelRemainingCr->setText("");
            ui->bTakeCredit->setEnabled(false);
        }
        else if (ui->listWidget->currentItem()->text() == QString::number(vip.getId()))
        {
            vip.print(ui->labelName);
            ui->labelBalance->setText(QString::number(vip.getBalance()));
            ui->labelBills->setText(QString::number(vip.getBills()));
            ui->labelCredit->setText(QString::number(vip.getCredit()));
            ui->labelCreditBalance->setText(QString::number(vip.getCreditBalance()));
            ui->bTakeCredit->setEnabled(true);
```

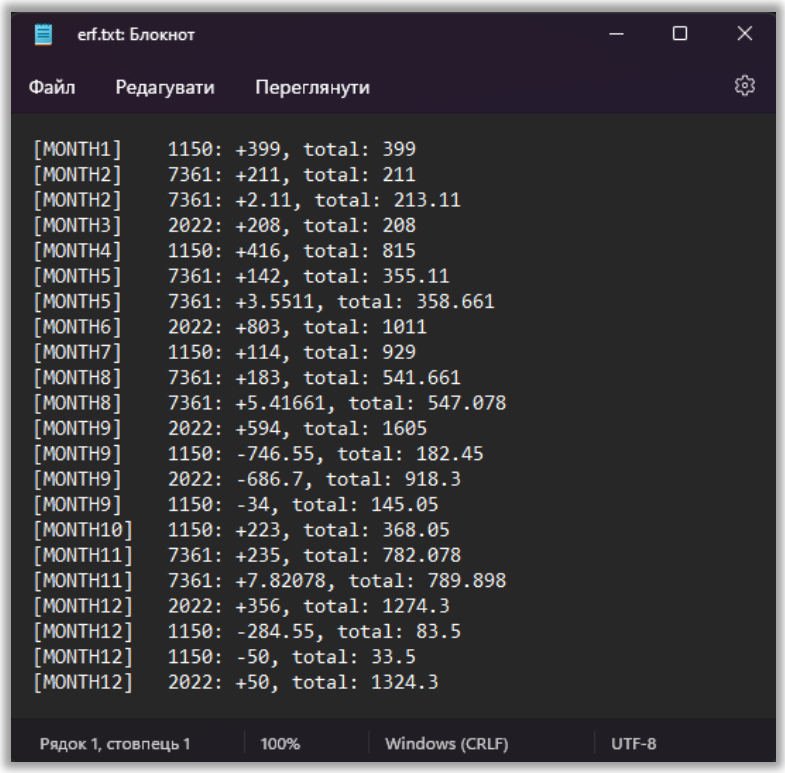
```
            ui->labelRemainingCr->setText(QString::number(VIPBankAccount::getCreditLimit() -
vip.getCreditTaken()));
        }
    }
}
void MainWindow::on_bPayBills_clicked()
{
    bool successful = false;
    if(ui->listWidget->currentItem()->text() == QString::number(common.getId()))
    {
        successful = common.PayBills();
    }
    else if (ui->listWidget->currentItem()->text() == QString::number(social.getId()))
    {
        successful = social.PayBills();
    }
    else if (ui->listWidget->currentItem()->text() == QString::number(vip.getId()))
    {
        successful = vip.PayBills();
    }
    if(!successful)
    {
        errorMessage("Paying bills not successful");
        return;
    }
    updateData();
}

void MainWindow::on_listWidget_itemSelectionChanged()
{
    ui->bPayBills->setEnabled(true);
    ui->bTransfer->setEnabled(true);
    ui->bWithdraw->setEnabled(true);
    ui->lineID->setEnabled(true);
    ui->lineMoney->setEnabled(true);
    updateData();
}


void MainWindow::on_bWithdraw_clicked()
{
    bool successful = false;
    if(!ui->lineMoney->text().isEmpty())
    {
       if(ui->listWidget->currentItem()->text() == QString::number(common.getId()))
       {
            successful = common.Withdraw(ui->lineMoney->text().toDouble());
       }
       else if (ui->listWidget->currentItem()->text() == QString::number(social.getId()))
       {
            successful = social.Withdraw(ui->lineMoney->text().toDouble());
       }
       else if (ui->listWidget->currentItem()->text() == QString::number(vip.getId()))
       {
           successful = vip.Withdraw(ui->lineMoney->text().toDouble());
       }
    }
    if(!successful)
    {
        errorMessage("Withdrawing money not successful");
        return;
    }
    updateData();
}


void MainWindow::on_bTransfer_clicked()
{
```

```
    bool successful = false;
    BankAccount *receiver = nullptr;
    if(!ui->lineID->text().isEmpty())
    {
        if(ui->lineID->text() == QString::number(common.getId()))
            receiver = &common;
        else if(ui->lineID->text() == QString::number(social.getId()))
            receiver = &social;
        else if(ui->lineID->text() == QString::number(vip.getId()))
            receiver = &vip;
    }
    if(!ui->lineMoney->text().isEmpty() && receiver)
    {
        if(ui->listWidget->currentItem()->text() == QString::number(common.getId()))
        {
             successful = common.TransferTo(ui->lineMoney->text().toDouble(), receiver);
        }
        else if (ui->listWidget->currentItem()->text() == QString::number(social.getId()))
        {
             successful = social.TransferTo(ui->lineMoney->text().toDouble(), receiver);
        }
        else if(ui->listWidget->currentItem()->text() == QString::number(vip.getId()))
        {
             successful = vip.TransferTo(ui->lineMoney->text().toDouble(), receiver);
        }
    }
    if(!successful)
    {
        errorMessage("Transfering money not successful");
        return;
    }
    updateData();
}


void MainWindow::on_bSkipMonth_clicked()
{
    common.SkipMonth();
    social.SkipMonth();
    vip.SkipMonth();
    updateData();
}


void MainWindow::on_bTakeCredit_clicked()
{
    if(!ui->lineMoney->text().isEmpty())
    {
        bool successful = false;
        if(!(successful = vip.takeCredit(ui->lineMoney->text().toDouble())))
        {
            errorMessage("Taking credit not successful");
            return;
        }
        updateData();
    }
}



void MainWindow::on_actionGeneral_history_triggered()
{
    BankAccount::printLog(ui->textEdit);
}


void MainWindow::on_actionHistory_for_common_account_triggered()
{
```

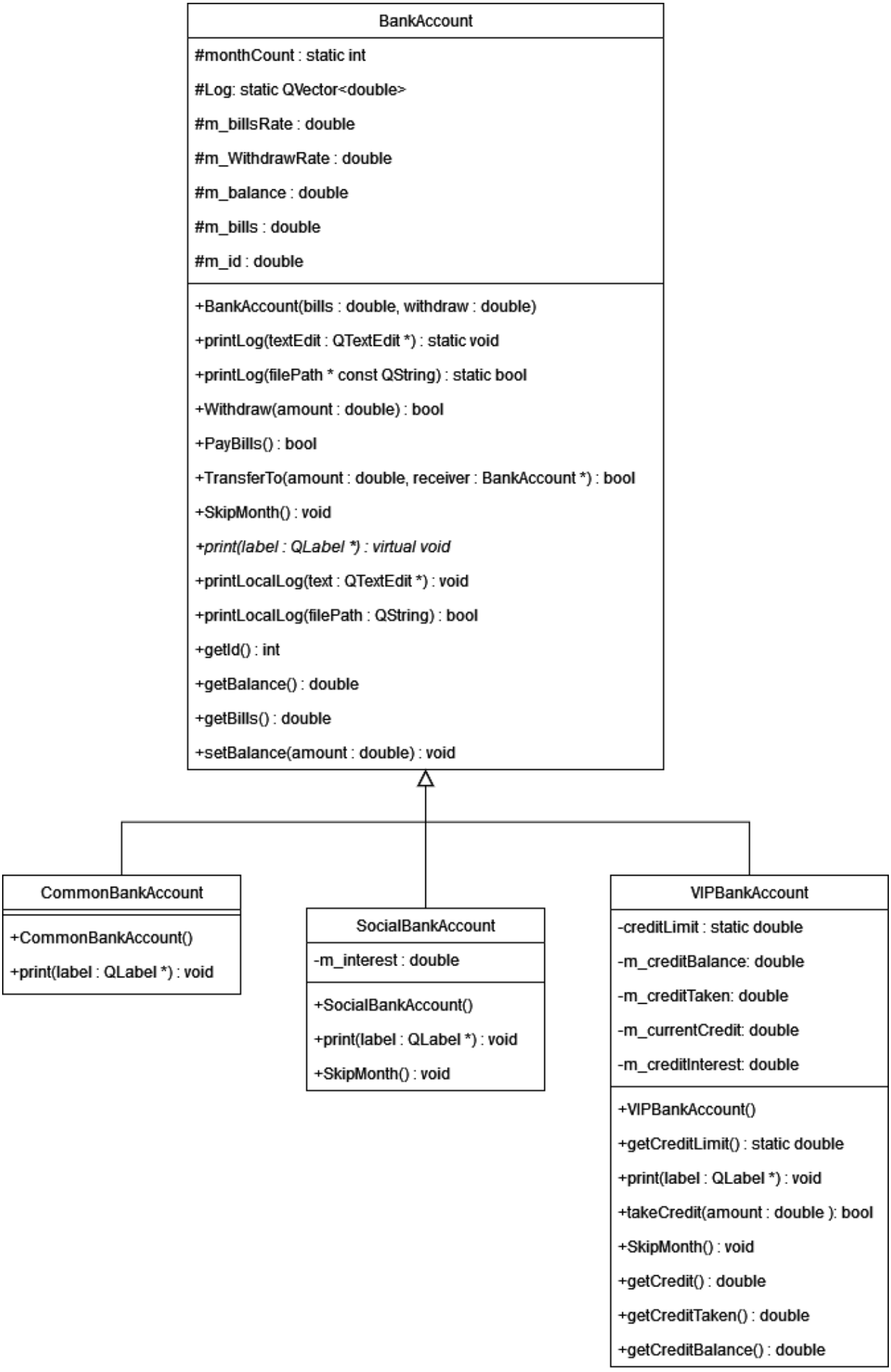
```
    common.printLocalLog(ui->textEdit);
}


void MainWindow::on_actionHistory_for_social_account_triggered()
{
    social.printLocalLog(ui->textEdit);
}


void MainWindow::on_actionHistory_for_VIP_account_triggered()
{
    vip.printLocalLog(ui->textEdit);
}


void MainWindow::on_actionGeneral_history_2_triggered()
{
    QString fileName = QFileDialog::getOpenFileName(this, "Please choose file",
QDir::currentPath(), tr("Text documents (*.txt)"));
    if(!BankAccount::printLog(fileName))
        errorMessage("Error writing to file");
}


void MainWindow::on_actionHistory_for_common_account_2_triggered()
{
    QString fileName = QFileDialog::getOpenFileName(this, "Please choose file",
QDir::currentPath(), tr("Text documents (*.txt)"));
    if(!common.printLocalLog(fileName))
        errorMessage("Error writing to file");
}


void MainWindow::on_actionHistory_for_social_account_2_triggered()
{
    QString fileName = QFileDialog::getOpenFileName(this, "Please choose file",
QDir::currentPath(), tr("Text documents (*.txt)"));
    if(!social.printLocalLog(fileName))
        errorMessage("Error writing to file");
}


void MainWindow::on_actionHistory_for_VIP_account_2_triggered()
{
    QString fileName = QFileDialog::getOpenFileName(this, "Please choose file",
QDir::currentPath(), tr("TXT, (*.txt)"));
    if(!vip.printLocalLog(fileName))
        errorMessage("Error writing to file");
}
```



Зображення графічного інтерфейсу програми



Зображення історії транзакцій



Діаграма класів

Висновок: 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.