МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **ІКНІ**
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:
гр. ПЗ-15
Гук М.М.

Прийняла:
доцент кафедри ПЗ
Яцишин С.І

« ____ » _____ 2020 р.

∑= ____ .....

Львів - 2020

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.

Індивідуальне 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 функцій в базовому класі, перевизначити їх в похідних.
6. Включити в 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.

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. 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.

Код програми

```
/*
 * Filename: account.h
 */
#ifndef ACCOUNT_H
#define ACCOUNT_H
#include <QFile>
#include <QMessageBox>
#include <QTextStream>
#include <QDialog>
#include "QStringList"
```

```
#include "bankaccount.h"
#include "bankdefault.h"
#include "bankspecial.h"
#include "municipal.h"
#include "credit.h"
#include "recharge.h"
#include "register.h"
#include "transfer.h"
#include "withdraw.h"
#include "bankvip.h"

namespace Ui {
class Account;
}

class Account : public QDialog
{
    Q_OBJECT

public:
    explicit Account(QWidget *parent = nullptr);

    ~Account();

private slots:
    void on_municipal_clicked();

    void on_transfer_clicked();

    void on_withdraw_clicked();

    void on_recharge_clicked();

    void on_credit_clicked();

    void on_transactions_clicked();

    void on_quit_clicked();
protected:
    BankAccount * basic;
private:
    QString clientInfo;
    TypeAccount type;
    QString typeStr;
    bool readCurrentClient();
    void defineType();
    void setBaseInfo();
    void createDefault();
    void createSpecial();
    void createVIP();
    void writeInfo() const;
    Ui::Account *ui;
};
```

```
#endif // ACCOUNT_H

/*
 * Filename: account.cpp
 */
#include "account.h"
#include "ui_account.h"
#include "bankaccount.h"

Account::Account(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Account)
{
    ui->setupUi(this);
    if(!readCurrentClient())
    {
        QMessageBox::warning(this, "Failure", "Cannot reach data");
        close();
    }
    defineType();

    switch(type)
    {
    case Default:
        createDefault();
        break;
    case Special:
        createSpecial();
        break;
    case VIP:
        createVIP();
        break;
    default:
        QMessageBox::warning(this, "Failure", "Undefined type");
        close();
    }
    setBaseInfo();
    writeInfo();
}

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

bool Account::readCurrentClient()
{
    QFile current("current.txt");
    if(!current.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        QMessageBox::warning(this,"Warning", "Cannot reach data");
        return false;
    }

    QTextStream in(&current);
```

```
    clientInfo = in.readLine();
    return true;
}

void Account::defineType()
{
    typeStr = clientInfo.section(' ', BankAccount::Type, BankAccount::Type);
    if(typeStr == "1")
    {
        typeStr = "Default";
        type = Default;
    }
    else if(typeStr == "2")
    {
        typeStr = "Special";
        type = Special;
    }
    else if(typeStr == "3")
    {
        typeStr = "VIP";
        type = VIP;
    }
    else
        type = None;
}

void Account::createDefault()
{
    BankDefault * bankDefault = new BankDefault;
    basic = bankDefault;
}

void Account::createSpecial()
{
    BankSpecial * bankSpecial = new BankSpecial;
    basic = bankSpecial;
}

void Account::createVIP()
{
    BankVIP * bankVIP = new BankVIP;
    basic = bankVIP;
    QString credit = clientInfo.section(' ', BankAccount::Credit,
BankAccount::Credit);
    bankVIP->createCredit(credit.toDouble());
    ui->creditLine->setText(credit);
    ui->credit->setEnabled(true);
}

void Account::writeInfo() const
{
    ui->name->setText(basic->getName());
    ui->type->setText(typeStr);
    ui->number->setText(QString::number(basic->getNum()));
    ui->balance->setText(QString::number(basic->getBalance()));
```

```
}

void Account::setBaseInfo()
{
    QString number = clientInfo.section(' ', BankAccount::Number,
BankAccount::Number);
    QString pin = clientInfo.section(' ', BankAccount::Pin, BankAccount::Pin);
    QString name = clientInfo.section(' ', BankAccount::Name, BankAccount::Name);
    QString balance = clientInfo.section(' ', BankAccount::Balance,
BankAccount::Balance);
    QString municipal = clientInfo.section(' ', BankAccount::Municipal,
BankAccount::Municipal);
    basic->setName(name);
    basic->setBalance(balance.toDouble());
    basic->setNum(number.toInt());
    basic->setPin(pin.toInt());
    basic->setMunicipal(municipal.toDouble());
}

void Account::on_municipal_clicked()
{
    Municipal municipalDialog;
    int commission = basic->getCommission();
    double municipal = basic->getMunicipal();
    municipalDialog.setValues(commission, municipal);
    municipalDialog.setModal(true);
    municipalDialog.exec();
    if(!municipalDialog.result())
        return;
    if(!basic->payBills())
        QMessageBox::warning(this, "Error", "Not enough money");
    else
        QMessageBox::information(this, "Success", "Municipal is paid");
    writeInfo();

}

void Account::on_transfer_clicked()
{
    Transfer transferDialog;

    int commission = basic->getCommission();
    transferDialog.setCommission(commission);
    transferDialog.setModal(true);
    transferDialog.exec();
    if(!transferDialog.result())
        return;
    int receiver = transferDialog.getReceiver();
    double sum = transferDialog.getSum();
    if(!basic->withdraw(sum))
    {
        QMessageBox::warning(this, "Error", "Not enough money");
        return;
    }
    if(!basic->transfer(sum, receiver))
    {
```

```
        QMessageBox::warning(this, "Error", "Can't transfer money");
        double total = transferDialog.getTotal();
        basic->recharge(total);
        return;
    }
    else
        QMessageBox::information(this, "Success", "Transfer is successful");

    writeInfo();

}

void Account::on_withdraw_clicked()
{
    Withdraw withdrawDialog;
    int commission = basic->getCommission();
    withdrawDialog.setCommission(commission);
    withdrawDialog.setModal(true);
    withdrawDialog.exec();
    if(!withdrawDialog.result())
        return;
    double sum = withdrawDialog.getSum();
    if(!basic->withdraw(sum))
        QMessageBox::warning(this, "Error", "Not enough money");
    else
        QMessageBox::information(this, "Success", "Withdraw is successful");
    writeInfo();
}

void Account::on_recharge_clicked()
{
    Recharge rechargeDialog;
    rechargeDialog.setModal(true);
    rechargeDialog.exec();
    if(!rechargeDialog.result())
        return;
    double sum = rechargeDialog.getSum();
    basic->recharge(sum);
    QMessageBox::information(this, "Success", "Recharge is successful");
    writeInfo();
}

void Account::on_credit_clicked()
{
    Credit creditDialog;
    creditDialog.setModal(true);
    creditDialog.exec();
    if(!creditDialog.result())
        return;
    double credit = creditDialog.getSum();
    ui->creditLine->setText(QString::number(credit));
    basic->recharge(credit);
    QMessageBox::information(this, "Success", "Credit is created");
    writeInfo();
}
```

```
void Account::on_transactions_clicked()
{
    QLinkedList<BankAccount::Transaction> history = basic->getTransactions();
    QString list = "QString list = "Receiver and sum:\n";;
    BankAccount::Transaction temp;
    while(!history.isEmpty())
    {
        temp = history.takeFirst();
        list += QString::number(temp.receiver) + ' ' + QString::number(temp.sum) + '\
n';
    }
    QMessageBox::information(this, "Transactions", list);
}

void Account::on_quit_clicked()
{
    QFile data("data.txt");
    if(!data.open(QIODevice::ReadWrite | QIODevice::Text))
    {
        QMessageBox::information(this, "Failure", "Data is not saved");
        close();
    }

    QTextStream out(&data);
    int num = basic->getNum();
    int pin = basic->getPin();
    QString name = basic->getName();
    double balance = basic->getBalance();
    double credit = ui->creditLine->text().toDouble();
    double municipal = basic->getMunicipal();
    QString newInfo = makeClient(num, pin, type, name,
                          balance, credit, municipal);
    refreshDataAboutClient(newInfo, clientInfo, out);
    QFile current("current.txt");
    current.open(QIODevice::WriteOnly | QIODevice::Text);
    current.close();
    data.close();
    close();
}

/*
 * Filename: bankaccount.h
 */
#ifndef BANKACCOUNT_H
#define BANKACCOUNT_H
#include <QMainWindow>
#include <QLinkedList>
#include "customer.h"
#define NUMBER_LEN 10
#define PIN_LEN 4
#define NAME_LEN 10
#define TYPES 3

class BankAccount : protected Customer
{
public:
```

```
    typedef struct
    {
        int receiver;
        double sum;
    } Transaction;
private:
    QLinkedList <Transaction> transactionList;
    int accNum;
    double balance;
    int PIN;
    int commission;
    double municipal;
protected:
    void setCommission(int percent);
public:
    enum Value {Number, Pin, Type, Name, Balance, Credit, Municipal, Values};
    BankAccount();
    BankAccount(int accNum, int PIN, QString name);
    using Customer::getName;
    using Customer::setName;
    virtual void recharge(double sum);
    virtual bool withdraw(double sum) = 0;
    bool transfer(double sum, int receiver);
    double getBalance() const;
    void setBalance(double sum);
    int getNum() const;
    void setNum(int accNum);
    int getPin() const;
    void setPin(int pin);
    void storeTransaction(double sum, int receiver);
    double getCommission() const;
    void setMunicipal(double sum);
    double getMunicipal() const;
    bool payBills();
    QLinkedList<Transaction> getTransactions() const;
};

#endif // BANKACCOUNT_H

/*
 * Filename: bankaccount.cpp
 */
#include "bankaccount.h"
#include "iofunctions.h"
#include <QFile>
#include <QTextStream>

BankAccount::BankAccount() : Customer()
{
    accNum = 0;
    balance = 0;
    PIN = 0;
    commission = 0;
    municipal = 0;
}
```

```
BankAccount::BankAccount(int newNumber, int newPIN, QString newName) :
Customer(newName)
{
    accNum = newNumber;
    balance = 0;
    PIN = newPIN;
    commission = 0;
    municipal = 0;
}
bool BankAccount::transfer(double sum, int receiver)
{
    QString receiverInfo;
    QString clientInfo;
    findClientInData(clientInfo, QString::number(accNum));
    bool receiverExist = findClientInData(receiverInfo, QString::number(receiver));
    QString oldBalance = getValue(receiverInfo, BankAccount::Balance);
    double newBalance = oldBalance.toDouble() + sum;

    if(!receiverExist)
        return false;
    if(!rechargeInData(QString::number(receiver), newBalance))
        return false;
    storeTransaction(sum, receiver);

    return true;

}

void BankAccount::recharge(double sum)
{
    balance += sum;
}

bool BankAccount::withdraw(double sum)
{
    if(balance - sum < 0)
        return false;
    else
        balance -= sum;
    return true;
}
double BankAccount::getBalance() const
{
    return balance;
}

void BankAccount::setBalance(double sum)
{
    balance = sum;
}

int BankAccount::getNum() const
{
    return accNum;
}
```

```
void BankAccount::setNum(int number)
{
    accNum = number;
}

int BankAccount::getPin() const
{
    return PIN;
}

void BankAccount::setPin(int newPin)
{
    PIN = newPin;
}

void BankAccount::storeTransaction(double newSum, int newReceiver)
{
    Transaction newTransaction;
    newTransaction.sum = newSum;
    newTransaction.receiver = newReceiver;
    transactionList << newTransaction;

}
void BankAccount::setCommission(int percent)
{
    commission = percent;
}

double BankAccount::getCommission() const
{
    return commission;
}

void BankAccount::setMunicipal(double sum)
{
    municipal = sum;
}

double BankAccount::getMunicipal() const
{
    return municipal;
}

bool BankAccount::payBills()
{
    if(this->withdraw(municipal))
    {
        municipal = 0;
        return true;
    }
    return false;
}

QLinkedList<BankAccount::Transaction> BankAccount::getTransactions() const
{
```

```
    return transactionList;
}

/*
 * Filename: bankdefault.h
 */
#ifndef BANKDEFAULT_H
#define BANKDEFAULT_H
#include "bankaccount.h"

class BankDefault : public BankAccount
{
private:
    enum {Commission = 10};
public:
    BankDefault();
    BankDefault(int accNum, int PIN, QString name);
    virtual bool withdraw(double sum);
};

#endif // BANKDEFAULT_H

/*
 * Filename: bankdefault.cpp
 */
#include "bankdefault.h"

BankDefault::BankDefault() : BankAccount()
{
    setCommission(Commission);
}
BankDefault::BankDefault(int accNum, int PIN, QString name) : BankAccount(accNum,
PIN, name)
{
    setCommission(Commission);
}

bool BankDefault::withdraw(double sum)
{
    sum = sum * getCommission() / 100 + sum;
    double currentBalance = getBalance();
    if( currentBalance - sum < 0)
        return false;
    else setBalance(currentBalance - sum);
    return true;
}

/*
 * Filename: bankspecial.h
 */
#ifndef SPECIAL_H
#define SPECIAL_H
#include "bankaccount.h"

class BankSpecial : public BankAccount
```

```
{
private:
    enum {Extra = 5};
public:
    BankSpecial() : BankAccount() {}
    BankSpecial(int accNum, int PIN, QString name);
    virtual bool withdraw(double sum);

};

#endif // SPECIAL_H

/*
 * Filename: bankspecial.cpp
 */
#include "bankspecial.h"

BankSpecial::BankSpecial(int accNum, int PIN, QString name) : BankAccount(accNum,
PIN, name)
{
    setCommission(0);
}

bool BankSpecial::withdraw(double sum)
{
    double currentBalance = getBalance();
    if( currentBalance - sum < 0)
        return false;
    else setBalance(currentBalance - sum);
    return true;
}

/*
 * Filename: bankvip.h
 */
#ifndef BANKVIP_H
#define BANKVIP_H
#include "bankaccount.h"
const double standart = 5;
const double high = 8;
class BankVIP : public BankAccount
{
    double credit;
    double percent;
    enum {Commission = 8};
public:
    enum {Limit = 5000};
    BankVIP();
    BankVIP(int accNum, int PIN, QString name, double credit);
    virtual bool withdraw(double sum);
    double getPercent(double sum) const;
    double getCredit() const;
    bool createCredit(double sum);
};
```

```
#endif // BANKVIP_H

/*
 * Filename: bankvip.cpp
 */
#include "bankvip.h"

BankVIP::BankVIP() : BankAccount()
{
    percent = standart;
    credit = 0;
    setCommission(Commission);
}

BankVIP::BankVIP(int accNum, int PIN, QString name, double newCredit) :
    BankAccount(accNum, PIN, name)
{
    credit = newCredit;
    setCommission(Commission);
}

bool BankVIP::withdraw(double sum)
{
    sum = sum * getCommission() / 100 + sum;
    double currentBalance = getBalance();
    if( currentBalance - sum < 0)
        return false;
    else setBalance(currentBalance - sum);
    return true;
}

double BankVIP::getCredit() const
{
    return credit;
}

double BankVIP::getPercent(double sum) const
{
    if(sum > Limit)
        return high;
    return standart;
}

bool BankVIP::createCredit(double sum)
{
    if(credit)
        return false;
    percent = getPercent(sum);
    recharge(sum);
    return true;
}

/*
 * Filename: credit.h
 */
```

```
#ifndef CREDIT_H
#define CREDIT_H
#include <QDialog>

namespace Ui {
class Credit;
}

class Credit : public QDialog
{
    Q_OBJECT

public:
    explicit Credit(QWidget *parent = nullptr);
    ~Credit();
    double getSum();
private slots:
    void on_sum_valueChanged(double arg1);

    void on_getCredit_clicked();

    void on_cancel_clicked();

private:
    double sum;
    double percent;
    Ui::Credit *ui;
};

#endif // CREDIT_H

/*
 * Filename: credit.cpp
 */
#include "credit.h"
#include "ui_credit.h"
#include "login.h"
#include "bankvip.h"
Credit::Credit(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Credit)
{
    ui->setupUi(this);
}

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

double Credit::getSum()
{
    return sum;
}
```

```
void Credit::on_sum_valueChanged(double newSum)
{
    if(sum < BankVIP::Limit)
        percent = standart;
    else
        percent = high;
    sum = newSum;
}

void Credit::on_getCredit_clicked()
{
    accept();
}

void Credit::on_cancel_clicked()
{
    reject();
}

/*
 * Filename: customer.h
 */
#ifndef CUSTOMER_H
#define CUSTOMER_H
#include <QString>

class Customer
{
protected:
     QString name;
public:
    Customer();
    Customer(QString name);
    virtual QString getName() const;
    virtual void setName(QString name);
};

#endif // CUSTOMER_H

/*
 * Filename: customer.cpp
 */
#include "customer.h"

Customer::Customer()
{
    name = "No name";
}

Customer::Customer(QString newName)
{
    name = newName;
}
QString Customer::getName() const
{
```

```
    return name;
}

void Customer::setName(QString newName)
{
    name = newName;
}

/*
 * Filename: iofunction.h
 */
#ifndef FIOFUNCTIONS_H
#define FIOFUNCTIONS_H
#include <QString>
#include <QFile>
#include <QStringList>
#include <QTextStream>
#include "bankaccount.h"
enum TypeAccount {None, Default, Special, VIP};
bool findClientInData(QString & clientInfo, const QString & accNum);
bool rechargeInData(const QString & accNum, double newBalance);
void changeBalance(QString &clientInfo, QString newBalance);
int getPosition(QString & clientInfo, const QString & accNum, QTextStream & in);
int writeClientToData(const QString & name, const QString & pin, TypeAccount type);
QString makeClient(int num, int pin, TypeAccount type, const QString & name,
                   double balance = 0, double credit = 0, double municipal = 0);
void refreshDataAboutClient(const QString & newInfo, const QString &oldInfo,
QTextStream &in);
bool getLastNum(QString & last, QTextStream & in);
bool writeCurrent(const QString & info);
bool checkNum(const QString & num);
bool checkPin(const QString & pin);
QString getValue(const QString & info, BankAccount::Value val);
#endif // FIOFUNCTIONS_H

/*
 * Filename: iofunctions.cpp
 */
#include "iofunctions.h"
bool findClientInData(QString &clientInfo, const QString &accNum)
{
    bool clientExist = false;
    QFile data("data.txt");
    if(!data.open(QIODevice::ReadOnly | QIODevice::Text))
        return clientExist;
    QTextStream in(&data);
    int pos = getPosition(clientInfo, accNum, in);
    if(pos >= 0)
        clientExist = true;
    data.close();
    return clientExist;
}

bool rechargeInData(const QString &accNum, double sum)
{
    bool result = false;
```

```
    QFile data("data.txt");
    if(!data.open(QIODevice::ReadWrite | QIODevice::Text))
        return result;
    QTextStream in(&data);
    QString oldInfo;
    if(!findClientInData(oldInfo, accNum))
        return result;
    QString newInfo = oldInfo;
    changeBalance(newInfo, QString::number(sum));
    refreshDataAboutClient(newInfo, oldInfo, in);
    data.close();
    return true;
}
void changeBalance(QString &clientInfo, QString newBalance)
{
    QString oldBalance = getValue(clientInfo, BankAccount::Balance);
    int posOfBalance = clientInfo.indexOf(oldBalance, 15);
    clientInfo.replace(posOfBalance, newBalance.length(), newBalance);
}
// Returns -1, if no such position
int getPosition(QString & clientInfo, const QString & accNum, QTextStream & in)
{
    int pos = -1;
    while(!in.atEnd())
    {
        pos = in.pos();
        clientInfo = in.readLine();
        QString curNum = getValue(clientInfo, BankAccount::Number);
        if(curNum == accNum)
            return pos;
    }
    return -1;
}
void refreshDataAboutClient(const QString & newInfo, const QString &oldInfo,
QTextStream &in)
{
    QString allInfo = in.readAll();
    allInfo.replace(oldInfo, newInfo);
    in.seek(0);
    in << allInfo;
}

int writeClientToData(const QString &name, const QString &pin, TypeAccount type)
{
    int result = 0;
    QString number;
    QFile data("data.txt");
    if(!data.open(QIODevice::ReadWrite | QIODevice::Text))
        return result;
    QTextStream out(&data);
    if(!getLastNum(number, out))
        return result;
    if(!std::next_permutation(number.begin(), number.end() ))
        return result;
    result = number.toInt();
    QString newClient = makeClient(result, pin.toInt(), type, name);
    out.seek(data.size());
```

```
    out << newClient;
    data.close();
    return result;
}

bool writeCurrent(const QString & info)
{
    QFile current("current.txt");
    if(!current.open(QIODevice::WriteOnly | QIODevice::Text))
        return false;

    QTextStream out(&current);
    out << info;
    current.close();
    return true;
}
bool getLastNum(QString & last, QTextStream & in)
{
    in.seek(0);
    QString temp = in.readLine();
    while(!in.atEnd())
        temp = in.readLine();
    last = getValue(temp, BankAccount::Number);
    return !temp.isEmpty();
}

bool checkNum(const QString & num)
{
    if(num.isEmpty() || num.length() != NUMBER_LEN)
        return false;
    for(int i = 0; i < num.length(); ++i)
        if(!num.at(i).isNumber())
            return false;
    return true;
}

bool checkPin(const QString & pin)
{
    if(pin.isEmpty() || pin.length() != PIN_LEN)
        return false;
    for(int i = 0; i < pin.length(); ++i)
        if(!pin.at(i).isNumber())
            return false;
    return true;
}

QString getValue(const QString &info, BankAccount::Value val)
{
    return info.section(' ', val, val);
}

QString makeClient(int num, int pin, TypeAccount type, const QString &name,
                   double balance, double credit, double municipal)
{
    QString newClient = QString::number(num) + ' ' + QString::number(pin)
            + ' ' + QString::number(type) + ' ' + name + ' ' +
```

```
            QString::number(balance) + ' ' + QString::number(credit) + ' '
            + QString::number(municipal) + '\n';
    return newClient;
}

/*
 * Filename: login.h
 */
#ifndef LOGIN_H
#define LOGIN_H
#include "iofunctions.h"
#include <QDialog>
#include <QFile>
#include <QMessageBox>
#include <QDebug>
#include <QTextStream>
#include "bankaccount.h"
namespace Ui {
class Login;
}

class Login : public QDialog
{
    Q_OBJECT

public:

    explicit Login(QWidget *parent = nullptr);
    ~Login();

private slots:

    void on_number_textChanged(const QString &arg1);

    void on_pin_textChanged(const QString &arg1);

    void on_logIn_clicked();

    void on_cancel_clicked();

private:
    bool numIsCorrect;
    bool pinIsCorrect;
    bool readClient(QString number, QString pin);
    bool checkCorrectPIN(QString clientInfo, QString pin);
    Ui::Login *ui;
};

#endif // LOGIN_H

/*
 * Filename: login.cpp
 */
#include "login.h"
```

```
#include "ui_login.h"
#include "account.h"

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

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

bool Login::readClient(QString number, QString password)
{
    QString clientInfo;
    bool result = false;
    bool clientExist = findClientInData(clientInfo, number);
    bool correctPassword = checkCorrectPIN(clientInfo, password);
    if(clientExist)
    {
        if(correctPassword)
            result = writeCurrent(clientInfo);
        else
            QMessageBox::warning(this,"Warning", "Incorrect password");
    }
    else
         QMessageBox::warning(this,"Warning", "Client doesn't exist");
    return result;
}

bool Login::checkCorrectPIN(QString clientInfo, QString pin)
{
    QString currentPIN = getValue(clientInfo, BankAccount::Pin);
    return currentPIN == pin;
}

void Login::on_number_textChanged(const QString &arg1)
{
    numIsCorrect = checkNum(arg1);
}

void Login::on_pin_textChanged(const QString &arg1)
{
    pinIsCorrect = checkPin(arg1);
}

void Login::on_logIn_clicked()
{
    if(!numIsCorrect || !pinIsCorrect)
    {
        QMessageBox::information(this,"Failure", "Fields are filled incorrectly");
        return;
```

```
    }
    QString pin = ui->pin->text();
    QString number = ui->number->text();

    if(readClient(number, pin))
    {
        QMessageBox::information(this, "Logged in", "Welcome!");
        Account * accountDialog = new Account;
        accountDialog->setModal(true);
        accountDialog->exec();
    }
    else
        QMessageBox::information(this,"Failure", "Impossible to log in");
    return;
}

void Login::on_cancel_clicked()
{
    close();
}

/*
 * Filename: mainwindow.h
 */
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
private slots:
    void on_logIn_clicked();

    void on_reg_clicked();

private:
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H

/*
 * Filename: mainwindow.cpp
 */
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "login.h"
```

```
#include "account.h"
#include "register.h"
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

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

void MainWindow::on_logIn_clicked()
{
    Login *loginDialog = new Login;
    loginDialog->setModal(true);
    loginDialog->exec();
}

void MainWindow::on_reg_clicked()
{
    Register *registerDialog = new Register;
    registerDialog->setModal(true);
    registerDialog->exec();
}

/*
 * Filename: municipal.h
 */
#ifndef MUNICIPAL_H
#define MUNICIPAL_H
#include <QDialog>

namespace Ui {
class Municipal;
}

class Municipal : public QDialog
{
    Q_OBJECT

public:
    explicit Municipal(QWidget *parent = nullptr);
    void setValues(double curCommission, double curMunicipal);
    ~Municipal();

private slots:
    void on_pay_clicked();

    void on_cancel_clicked();

private:
```

```
    double commission;
    double municipal;

    Ui::Municipal *ui;
};

#endif // MUNICIPAL_H

/*
 * Filename: municipal.cpp
 */
#include "municipal.h"
#include "ui_municipal.h"

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

void Municipal::setValues(double curCommission, double curMunicipal)
{
    commission = curCommission / 100;
    municipal = curMunicipal;
    ui->check->setText(QString::number(municipal));
    ui->commission->setText(QString::number(commission * municipal));
    ui->total->setText(QString::number(commission * municipal + municipal));
}

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

void Municipal::on_pay_clicked()
{
    accept();
}

void Municipal::on_cancel_clicked()
{
    reject();
}

/*
 * Filename: recharge.h
 */
#ifndef RECHARGE_H
#define RECHARGE_H
#include <QDialog>

namespace Ui {
class Recharge;
```

```
}

class Recharge : public QDialog
{
    Q_OBJECT

public:
    explicit Recharge(QWidget *parent = nullptr);
    double getSum() const;
    ~Recharge();

private slots:
    void on_sum_valueChanged(double arg1);

    void on_recharge_clicked();

    void on_cancel_clicked();

private:
    double sum;
    Ui::Recharge *ui;
};

#endif // RECHARGE_H

/*
 * Filename: recharge.cpp
 */
#include "recharge.h"
#include "ui_recharge.h"

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

double Recharge::getSum() const
{
    return sum;
}

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

void Recharge::on_sum_valueChanged(double newSum)
{
    sum = newSum;
}

void Recharge::on_recharge_clicked()
```

```
{
    accept();
}

void Recharge::on_cancel_clicked()
{
    reject();
}

/*
 * Filename: register.h
 */
#ifndef REGISTER_H
#define REGISTER_H
#include "login.h"
#include "iofunctions.h"
#include <QDialog>
#include <QFile>
#include <QMessageBox>
#include <QTextStream>
#include "bankaccount.h"
#include <QDialog>

namespace Ui {
class Register;
}

class Register : public QDialog
{
    Q_OBJECT

public:
    explicit Register(QWidget *parent = nullptr);
    ~Register();

private slots:

    void on_reg_clicked();

    void on_cancel_clicked();

    void on_name_textChanged(const QString &arg1);

    void on_pin_textChanged(const QString &arg1);
    void typeChosen();

private:
    int accNum;
    bool nameIsCorrect;
    bool pinIsCorrect;
    bool typeIsChosen;

    int createAccount();
    TypeAccount defineAccountType();
```

```
    Ui::Register *ui;
};

#endif // REGISTER_H
bool checkName(QString name);

/*
 * Filename: register.cpp
 */
#include "register.h"
#include "ui_register.h"
#include <algorithm>
#include "bankaccount.h"
Register::Register(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Register)
{
    ui->setupUi(this);
    nameIsCorrect = pinIsCorrect = typeIsChosen = 0;
    QCheckBox * typeAccount[TYPES] = {ui->def, ui->vip, ui->special };
    for(int i = 0; i < TYPES; ++i)
        connect(typeAccount[i], SIGNAL(toggled(bool)), this, SLOT(typeChosen()));
}

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

void Register::on_reg_clicked()
{
    if(!nameIsCorrect || !pinIsCorrect || !typeIsChosen)
    {
        QMessageBox::information(this,"Failure", "Fields are filled incorrectly");
        return;
    }
    if(accNum = createAccount()) {
        QString msg = "Registration is successful.\n Your account number:" +
QString::number(accNum);
        QMessageBox::information(this,"Success", msg);
    }
    else
        QMessageBox::information(this,"Failure", "Registration is failed");
   close();
}

void Register::on_cancel_clicked()
{
    close();
}

bool checkName(QString name)
{
    if(name.isEmpty() || name.length() > NAME_LEN)
        return false;
    for(int i = 0; i < name.length(); ++i)
```

```
        if(!name.at(i).isLetterOrNumber())
            return false;
    return true;
}

void Register::on_name_textChanged(const QString &arg1)
{
    nameIsCorrect = checkName(arg1);
}

void Register::on_pin_textChanged(const QString &arg1)
{
    pinIsCorrect = checkPin(arg1);
}

void Register::typeChosen()
{
    typeIsChosen = true;
}

int Register::createAccount()
{
    TypeAccount typeAccount = defineAccountType();
    if(!typeAccount)
        return false;
    QString pin = ui->pin->text();
    QString name = ui->name->text();
    return writeClientToData(name, pin, typeAccount);
}

TypeAccount Register::defineAccountType()
{
    TypeAccount type = None;
    QCheckBox * typeAccount[TYPES] = {ui->def, ui->special, ui->vip};
    for(int i = 0 ; i < TYPES; ++i)
        if(typeAccount[i]->checkState())
        {
            switch(i + 1)
            {
            case Default:
                type = Default;
                break;
            case Special:
                type = Special;
                break;
            case VIP:
                type = VIP;
                break;
            }
        }
    return type;

}

/*
```

```
 * Filename: transfer.h
 */
#ifndef TRANSFER_H
#define TRANSFER_H
#include <QMessageBox>
#include <QDialog>
#include "iofunctions.h"

namespace Ui {
class Transfer;
}

class Transfer : public QDialog
{
    Q_OBJECT

public:
    explicit Transfer(QWidget *parent = nullptr);
    ~Transfer();
    int getReceiver() const;
    double getTotal() const;
    double getSum() const;
    void setCommission(int curCommission);

private slots:
    void on_cancel_clicked();

    void on_transfer_clicked();

    void on_sum_valueChanged(double arg1);

    void on_to_textChanged(const QString &arg1);

private:
    bool numIsCorrect;
    int receiver;
    double total;
    double sum;
    int commission;
    Ui::Transfer *ui;
};

#endif // TRANSFER_H

/*
 * Filename: transfer.cpp
 */
#include "transfer.h"
#include "ui_transfer.h"

Transfer::Transfer(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Transfer)
{
```

```
    ui->setupUi(this);
    numIsCorrect = false;
    receiver = 0;
}

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

int Transfer::getReceiver() const
{
    return receiver;
}

double Transfer::getTotal() const
{
    return total;
}

double Transfer::getSum() const
{
    return sum;
}

void Transfer::setCommission(int curCommission)
{
    commission = curCommission;
    ui->commission->setText(QString::number(commission));
}

void Transfer::on_cancel_clicked()
{
    reject();
}

void Transfer::on_transfer_clicked()
{
    if(numIsCorrect)
    {
        accept();
        return;
    }
    QMessageBox::warning(this, "Error", "Incorrect receiver");
}

void Transfer::on_sum_valueChanged(double newSum)
{
    sum = newSum;
    total = newSum * commission / 100 + newSum;
    ui->total->setText(QString::number(total));
}

void Transfer::on_to_textChanged(const QString &num)
{
```

```
    numIsCorrect = checkNum(num);
    if(numIsCorrect)
        receiver = num.toInt();
}

/*
 * Filename: withdraw.h
 */
#ifndef WITHDRAW_H
#define WITHDRAW_H
#include <QDialog>

namespace Ui {
class Withdraw;
}

class Withdraw : public QDialog
{
    Q_OBJECT

public:
    explicit Withdraw(QWidget *parent = nullptr);
    ~Withdraw();
    void setCommission(double curCommission);
    double getTotal() const;
    double getSum() const;

private slots:
    void on_withdraw_clicked();

    void on_cancel_clicked();

    void on_sum_valueChanged(double sum);

private:
    double total;
    double sum;
    double commission;
    Ui::Withdraw *ui;
};

#endif // WITHDRAW_H

/*
 * Filename: withdraw.cpp
 */
#include "withdraw.h"
#include "ui_withdraw.h"

Withdraw::Withdraw(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Withdraw)
{
    ui->setupUi(this);
```

```
}

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

void Withdraw::setCommission(double curCommission)
{
    commission = curCommission;
    ui->commission->setText(QString::number(commission));
}

double Withdraw::getTotal() const
{
    return total;
}

double Withdraw::getSum() const
{
    return sum;
}

void Withdraw::on_withdraw_clicked()
{
    accept();
}

void Withdraw::on_cancel_clicked()
{
    reject();
}

void Withdraw::on_sum_valueChanged(double newSum)
{
    sum = newSum;
    total = newSum * commission / 100 + newSum;
    ui->total->setText(QString::number(total));
}
```

UML-діаграми до програми

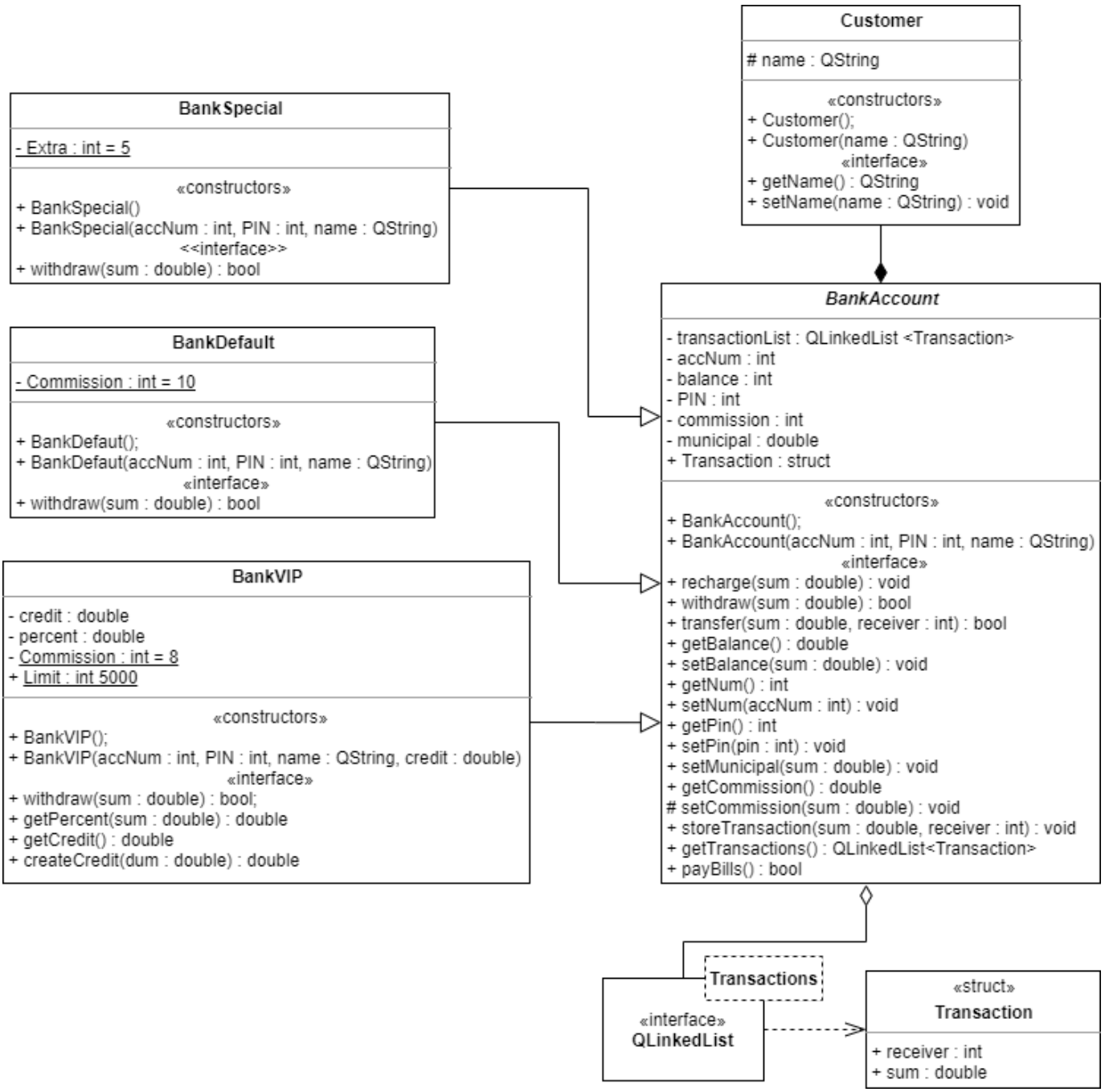

Рис 1: UML-діаграма програми

Протокол програми

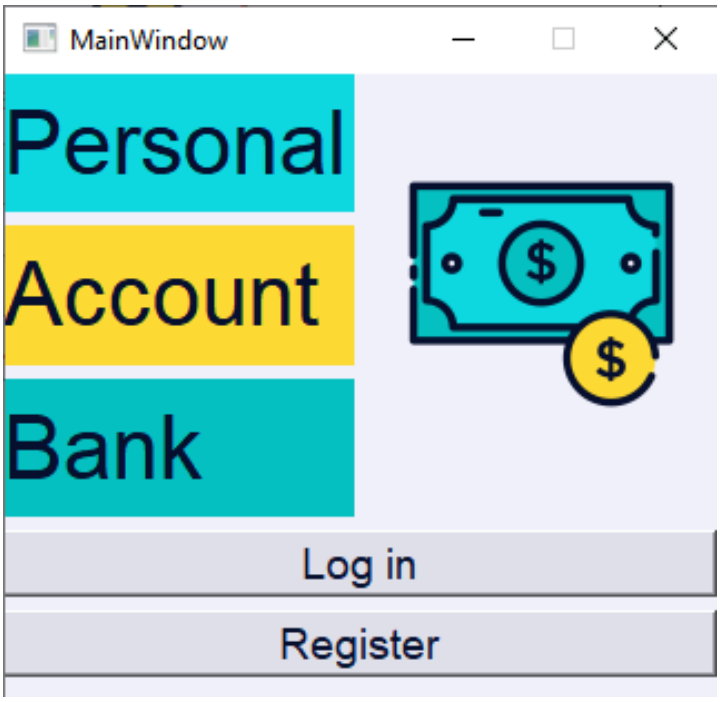

Рис 3: Головне меню програми

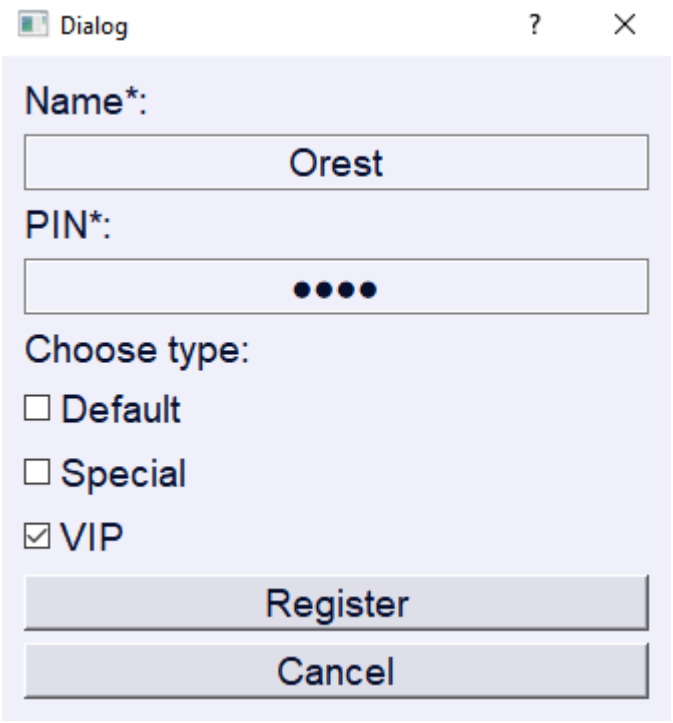

Рис 2: Реєстрація користувача Orest

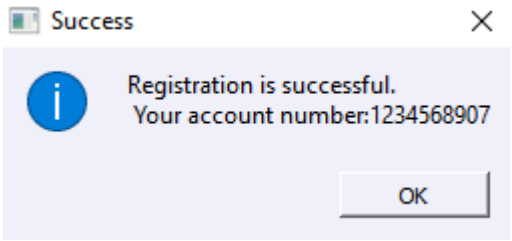

Рис 5: Повідомлення, при вдалій реєстрації

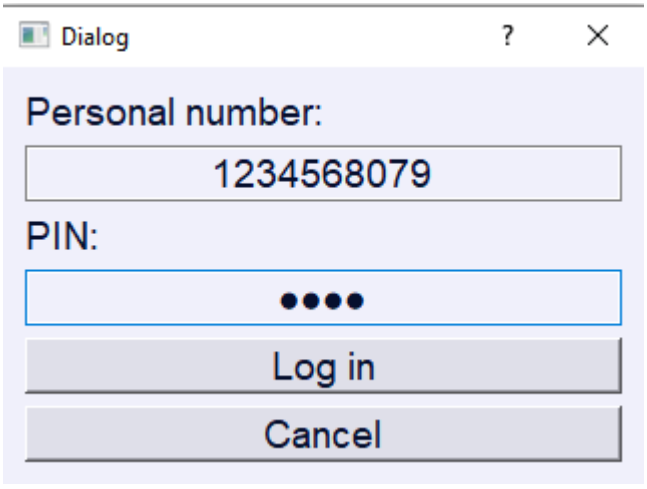

Рис 4: Вхід користувача

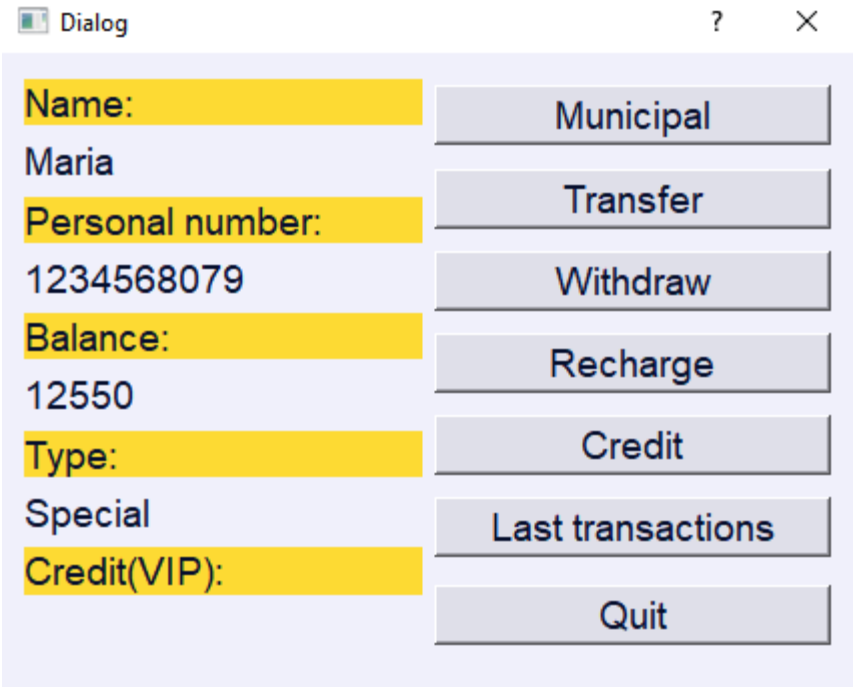

Рис 6: Меню користувача Maria

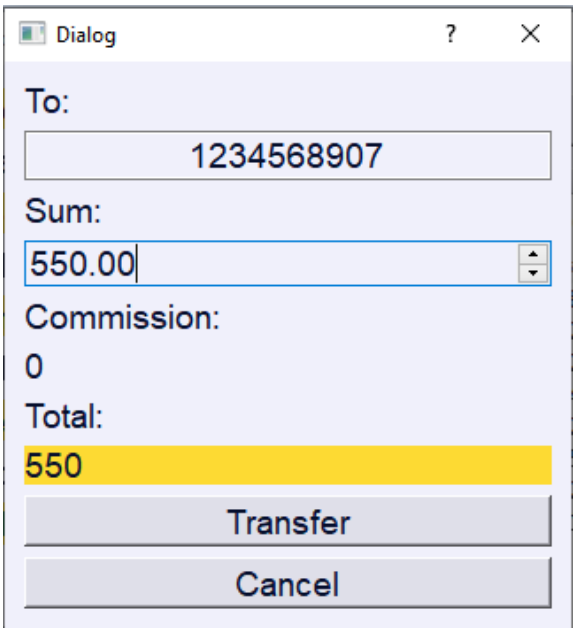

Рис 7: Переказання готівки користувачу Orest

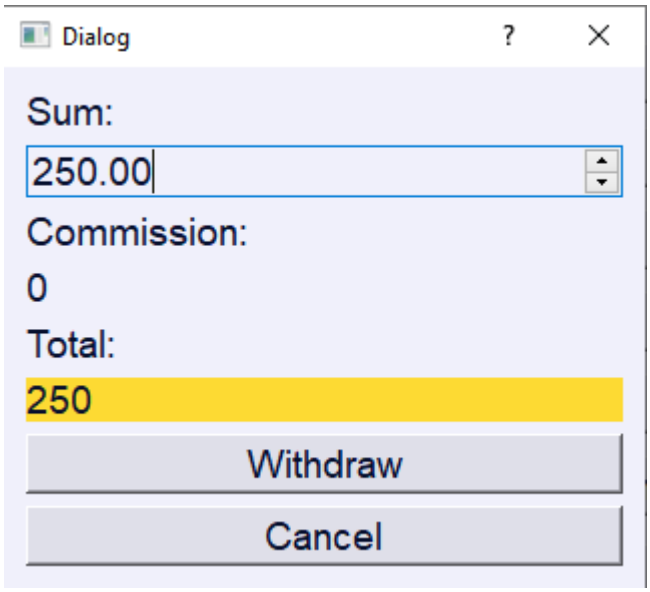

Рис 8: Зняття готівки

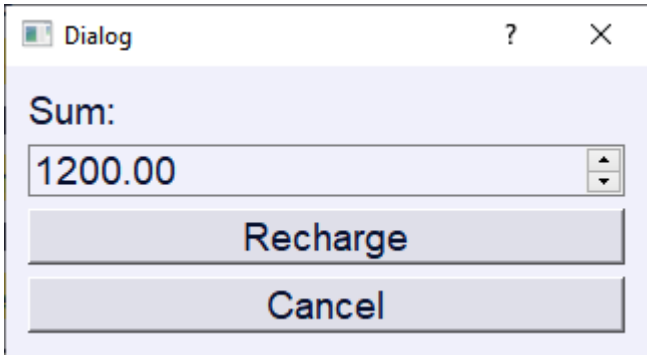

Рис 9: Поповнення балансу

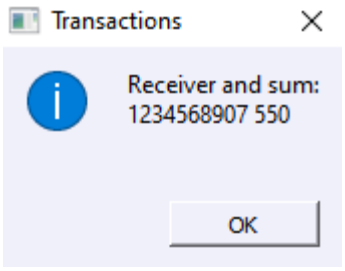

Рис 10: Список транзакцій

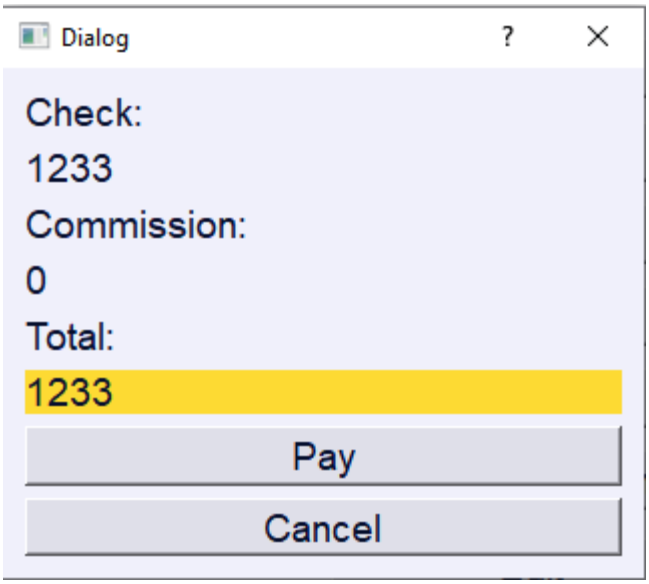

Рис 11: Сплата комунальних послуг

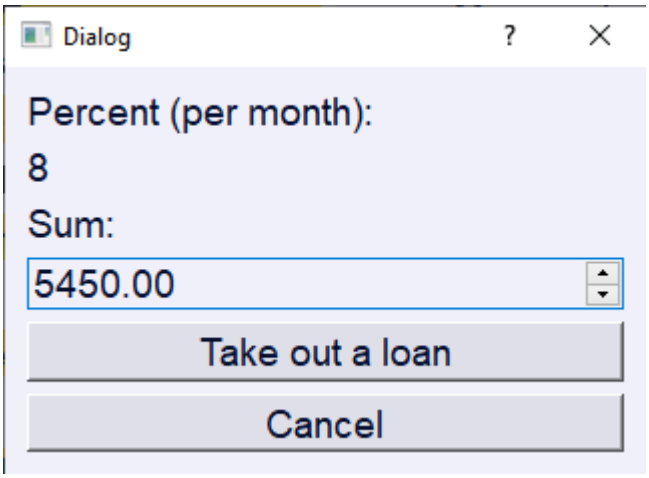

Рис 12: Оформлення кредиту

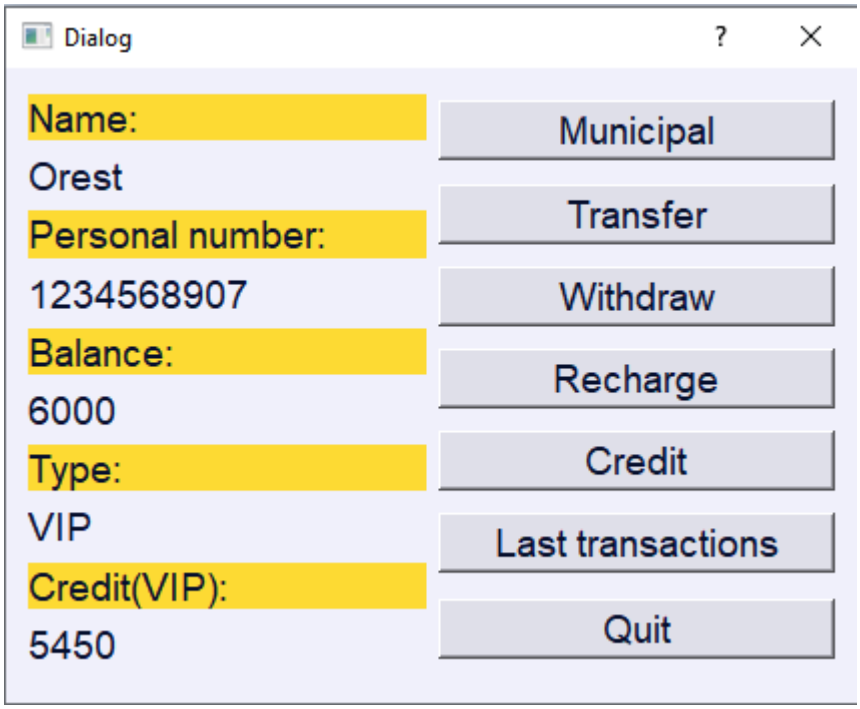

Рис 13: Меню користувача Orest

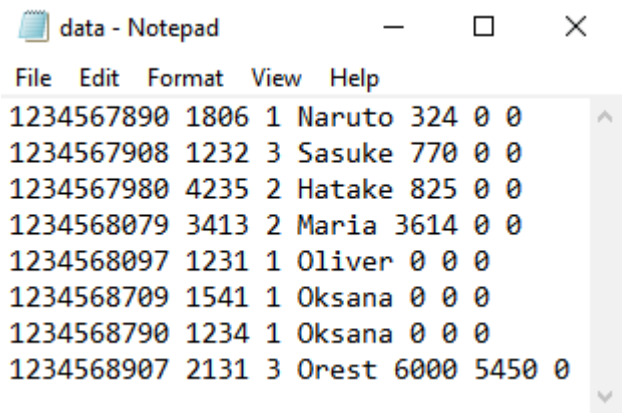

Рис 14: Файл із даними про користувачів

Висновок

На лабораторній роботі була розглянуте наслідування, створення та використання ієрархії класів. Я навчилась створювати базові та похідні класи, використовувати наслідування різного типу доступу, опанувавала принципи використання множинного наслідування. Зрозуміла, як перевизначати методи в похідному класі, освоїла принципи такого перевизначення.