Міністерство освіти і науки України
Національний університет “Львівська політехніка”
Інститут комп’ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

Звіт
Про виконання лабораторних робіт №8

На тему:

«Наслідування. Створення та використання ієрархії класів.»

Лектор:
доцент каф. ПЗ
Коротєєва Т.О.
Виконав:
ст. гр. ПЗ-11
Морозов О. Р.
Прийняла:
доцент каф. ПЗ
Коротєєва Т.О.

« __ » _____ 2022 р.

∑ = _____ _____ .

Львів – 2022

Тема: Наслідування. Створення та використання ієрархії класів.
Мета: Навчитися створювати базові та похідні класи, використовувати наслідування різного типу доступу, опанувати принципи використання множинного наслідування. Навчитися 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.

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.

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 класу Dog має доступ до всіх функцій-членів класу Mammal 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.

В заміщенні функції ми викликаємо базовий варіант цієї функції і розширюємо його додатковим функціоналом.

Або ж ми можемо викликати цей метод із базового класу. Для цього необхідно вказати область видимості базового класу.

Множинне наслідування.

В мові програмування c++ є можливість множинного наслідування. Приклад використання такого наслідування:

```
class Animal
{
private:
    int age;
public:
    int GetAge(int a) {
        return age;
    }
};

class Horse : public Animal {
public:
    virtual void Gallop(){ cout << "Gallop !!!"; }
};

class Bird : public Animal {
public:
    virtual void Fly(){ cout << "Fly"; }
};

class Pegasus : public Horse, public Bird {
public:
    void Chirp(){ cout << "Whinny!"; }
};
```

При створення об’єкта класу Pegasus будуть викликатись конструктори класів Horse і Bird саме в такій послідовності. Вкінці викличеться конструктор класу Pegasus.

Параметр рівня доступу при наслідуванні

При наслідуванні члени базового класу стають членами похідного класу. Як правило, для наслідування використовується наступна синтаксична конструкція.

```
class імя_похідного_класу: рівень_доступу імя_базового_класу
{
// тіло класу
}
```

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 ДО 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.

Завдання згідно варіанту

Варіант 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, нараховуєтсья 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.

Код програми

bankaccount.h

```
#ifndef BANKACCOUNT_H
#define BANKACCOUNT_H

#ifndef SET_H
#define SET_H
#include <QString>
#include <QLabel>
#include <QTableWidget>
#include <QLineEdit>
#include <QTextStream>
#include <QFile>
#endif //SET_H

class BankAccount
{
public:
 double m_money;
 QString m_transactionHistory;

 BankAccount(){};

 BankAccount(double m){
  m_money = m;
  m_transactionHistory = "";
 }

 ~BankAccount(){}
 void withdrawMoney(double amount){};
 void transferMoney(double amount){};
 void payForUtilities(double amount){};
 void displayTransactionHistory(QTableWidget *tbw, double money){};
 void enterOnAccount(double amount){};
 void saveInFile(QString str){};
 QString print(){
  return "This class: BankAccount. All classes StandartBankAccount, SocialBankAccount,
VIPBankAccount";
 }

};

class StandartBankAccount : protected BankAccount{
public:
 StandartBankAccount(double m):BankAccount(m){}
```

```
 double getMoney();
 void withdrawMoney(double amount);
 void transferMoney(double amount);
 void payForUtilities(double amount);
 void displayTransactionHistory(QTableWidget *tbw, double money);
 void enterOnAccount(double amount);
 void saveInFile(QString str);
 QString print();
};

class SocialBankAccount : public BankAccount{

public:
 SocialBankAccount(double m):BankAccount(m){}

 void withdrawMoney(double amount);
 void transferMoney(double amount);
 void payForUtilities(double amount);
 void displayTransactionHistory(QTableWidget *tbw, double money);
 void enterOnAccount(double amount);
 void saveInFile(QString str);
 QString print();

 void defoultDeposit();
};

class VIPBankAccount : public BankAccount{

protected:
 double m_creditMoney;

public:
 VIPBankAccount(double m):BankAccount(m){m_creditMoney = 0;}

 void withdrawMoney(double amount);
 void transferMoney(double amount);
 void payForUtilities(double amount);
 void displayTransactionHistory(QTableWidget *tbw, double money);
 void enterOnAccount(double amount);
 void saveInFile(QString str);
 QString print();

 void defoultDeposit();
 void takeCredit(double creditMoney);
 void creditPercents();

 double getCreditMoney();
 void returnCredit();
};

#endif // BANKACCOUNT_H
```

bankaccount.cpp

```
#include "bankaccount.h"
```

```
#define STANDART_COMMISION 0.03
#define SOCIAL_DEPOSIT_PERCENT (0.05)

#define VIP_CREDIT_LIMIT 100000
#define VIP_DEPOSIT_PERCENT 0.07
#define VIP_CREDIT_PERCENT 0.1

//-------------------------------------------------------------------------------
//---------------------------------Standart Bank Account-----------------------------
//-------------------------------------------------------------------------------

double StandartBankAccount::getMoney()
{
return m_money;
}

void StandartBankAccount::withdrawMoney(double amount) {
 m_money -=amount + (amount * STANDART_COMMISION);

 QString str = "Зі звичайного рахунку знято " + QString::number(amount) + "грн" + " + "
+ QString::number(amount * STANDART_COMMISION) + "грн комісії" +
            "\nНа рахунку залишилось " + QString::number(m_money) + "грн.";
 saveInFile(str);
}

void StandartBankAccount::transferMoney(double amount)
{
 m_money -= amount + (amount * STANDART_COMMISION);

 QString str = "Зі звичайного рахунку знято " + QString::number(amount + (amount *
STANDART_COMMISION)) + "грн.\nНа рахунку залишилось " +
QString::number(m_money) + "грн." +
    "\nПереказано " + QString::number(amount) +"грн.";
 saveInFile(str);
}

void StandartBankAccount::payForUtilities(double amount)
{
 m_money -=amount;

 QString str = "Зі звичайного рахунку знято " + QString::number(amount) + "грн.\nНа
рахунку залишилось " + QString::number(m_money) + "грн.";
 saveInFile(str);
}

void StandartBankAccount::displayTransactionHistory(QTableWidget *tbw, double
money)
{
tbw->insertRow(0);
QTableWidgetItem *item = new QTableWidgetItem();
QString str = "0";
if(money>0){
   str = "+" + QString::number(money);
  } else {
```

```
   str = QString::number(money);
  }
item->setText(str);
tbw->setItem(0, 0, item);
m_transactionHistory += str += ", ";
}

void StandartBankAccount::enterOnAccount(double amount)
{
 amount = amount - amount*STANDART_COMMISION;
 m_money +=amount;
 QString str = "На звичайний рахунок додано " + QString::number(amount) + "грн.\nНа
рахунку наявно " + QString::number(m_money) + "грн.";
}

void StandartBankAccount::saveInFile(QString str) {
 QTextStream out(stdout);

 QFile file("D:\\University\\2_semester\\OOP\\Lab8\\Backup\\StandartAccoutData.txt");

 if(file.open(QIODevice::WriteOnly)){
    QTextStream out(&file);
    str+="\n\n";
    out << str << Qt::endl;
  } else {
    qWarning("Помилка: файл не можливо відкрити");
  }
 file.close();
}

QString StandartBankAccount::print()
{
 return "This class: StandartBankAccount. All classes StandartBankAccount,
SocialBankAccount, VIPBankAccount";
}

//-------------------------------------------------------------------------------
//--------------------------------VIP Bank Account--------------------------------
//-------------------------------------------------------------------------------

void VIPBankAccount::withdrawMoney(double amount)
{
 m_money -=amount;

 QString str = "З ВІП рахунку знято " + QString::number(amount) + "грн.\nНа рахунку
залишилось " + QString::number(m_money) + "грн.";
 saveInFile(str);
}

void VIPBankAccount::transferMoney(double amount)
{
 m_money -= amount;

 QString str = "Зі ВІП рахунку знято " + QString::number(amount) + "грн.\nНа рахунку
залишилось " + QString::number(m_money) + "грн." +
```

```
    "\nПереказано " + QString::number(amount) +"грн.";
 saveInFile(str);
}

void VIPBankAccount::payForUtilities(double amount)
{
 m_money -=amount;

 QString str = "З ВІП рахунку знято " + QString::number(amount) + "грн.\nНа рахунку
залишилось " + QString::number(m_money) + "грн.";
 saveInFile(str);
}

void VIPBankAccount::displayTransactionHistory(QTableWidget *tbw, double money)
{
 tbw->insertRow(0);
 QTableWidgetItem *item = new QTableWidgetItem();
 QString str = "0";
 if(money>0){
   str = "+" + QString::number(money);
  } else {
   str = QString::number(money);
  }
 item->setText(str);
 tbw->setItem(0, 0, item);
 m_transactionHistory += str += ", ";
}

void VIPBankAccount::enterOnAccount(double amount)
{
 m_money +=amount;
 QString str = "На ВІП рахунок додано " + QString::number(amount) + "грн.\nНа
рахунку наявно " + QString::number(m_money) + "грн.";
 saveInFile(str);
}

void VIPBankAccount::saveInFile(QString str)
{
 QTextStream out(stdout);

 QFile file("D:\\University\\2_semester\\OOP\\Lab8\\Backup\\VIPAccoutData.txt");

 if(file.open(QIODevice::Append)){
    QTextStream out(&file);
    str+="\n\n";
    out << str << Qt::endl;
  } else {
    qWarning("Помилка: файл не можливо відкрити");
  }
 file.close();
}

QString VIPBankAccount::print()
{
```

```
 return "This class: VIPBankAccount. All classes StandartBankAccount,
SocialBankAccount, VIPBankAccount";
}

void VIPBankAccount::defoultDeposit()
{
 if(m_money > 10000){
    m_money += m_money * VIP_DEPOSIT_PERCENT;
    QString str = "На ВІП рахунок нараховано депозит " +
QString::number(m_money*VIP_DEPOSIT_PERCENT) + "грн.\nНа рахунку наявно " +
QString::number(m_money) + "грн.\n\n";
    saveInFile(str);
   }
}

void VIPBankAccount::takeCredit(double creditMoney)
{
if(creditMoney <= VIP_CREDIT_LIMIT && creditMoney > 0){
    m_creditMoney += creditMoney;
    m_money += creditMoney;
QString str = "На ваш рахунок додано " + QString::number(m_creditMoney) + "грн
кредитних коштів.\nНа рахунку наявно " + QString::number(m_money) + "грн.";
saveInFile(str);
  }
}

void VIPBankAccount::creditPercents()
{
 m_money -= m_creditMoney * VIP_CREDIT_PERCENT;
 QString str = "З ВІП рахунку знято " + QString::number(m_creditMoney *
VIP_CREDIT_PERCENT) + "грн для погашення відсотків по кредиту.\nНа рахунку
залишилось " + QString::number(m_money) + "грн.";
 saveInFile(str);
}

double VIPBankAccount::getCreditMoney()
{
return m_creditMoney;
}

void VIPBankAccount::returnCredit()
{
m_money-= m_creditMoney;

QString str = "З ВІП рахунку знято " + QString::number(m_creditMoney) + "грн для
погашення кредитної заборгованості.\nНа рахунку залишилось " +
QString::number(m_money) + "грн.";
m_creditMoney = 0;
saveInFile(str);
}

//-------------------------------------------------------------------------------
//--------------------------------Social Bank Account-----------------------------
//-------------------------------------------------------------------------------
```

```
void SocialBankAccount::withdrawMoney(double amount)
{
 m_money -=amount;

 QString str = "З соціального рахунку знято " + QString::number(amount) + "грн.\nНа
рахунку залишилось " + QString::number(m_money) + "грн.";
 saveInFile(str);
}

void SocialBankAccount::transferMoney(double amount)
{
 m_money -= amount;

 QString str = "Зі соціального рахунку знято " + QString::number(amount) + "грн.\nНа
рахунку залишилось " + QString::number(m_money) + "грн." +
    "\nПереказано " + QString::number(amount) +"грн.";
 saveInFile(str);
}

void SocialBankAccount::payForUtilities(double amount)
{
 m_money -=amount;

 QString str = "Зі соціального рахунку знято " + QString::number(amount) + "грн.\nНа
рахунку залишилось " + QString::number(m_money) + "грн.";

 saveInFile(str);
}

void SocialBankAccount::displayTransactionHistory(QTableWidget *tbw, double
money)
{
 tbw->insertRow(0);
 QTableWidgetItem *item = new QTableWidgetItem();
 QString str = "0";
 if(money>0){
   str = "+" + QString::number(money);
  } else {
   str = QString::number(money);
  }
 item->setText(str);
 tbw->setItem(0, 0, item);
 m_transactionHistory += str += ", ";
}

void SocialBankAccount::enterOnAccount(double amount)
{
 m_money +=amount;
 QString str = "На соціальний рахунок додано " + QString::number(amount) + "грн.\nНа
рахунку наявно " + QString::number(m_money) + "грн.\n\n";
 saveInFile(str);
}

void SocialBankAccount::saveInFile(QString str)
{
```

```
 QTextStream out(stdout);

 QFile file("D:\\University\\2_semester\\OOP\\Lab8\\Backup\\SocialAccoutData.txt");

 if(file.open(QIODevice::Append)){
    QTextStream out(&file);
    str += "\n";
    out << str << Qt::endl;
  } else {
    qWarning("Помилка: файл не можливо відкрити");
  }
 file.close();
}

QString SocialBankAccount::print()
{
 return "This class: SocialBankAccount. All classes StandartBankAccount,
SocialBankAccount, VIPBankAccount";
}

void SocialBankAccount::defoultDeposit()
{
   m_money = (m_money * SOCIAL_DEPOSIT_PERCENT) + m_money;
   QString str = "На соціальний рахунок нараховано депозит " +
QString::number(m_money*SOCIAL_DEPOSIT_PERCENT) + "грн.\nНа рахунку наявно "
+ QString::number(m_money) + "грн.\n\n";
   saveInFile(str);
}
```

Вигляд програми

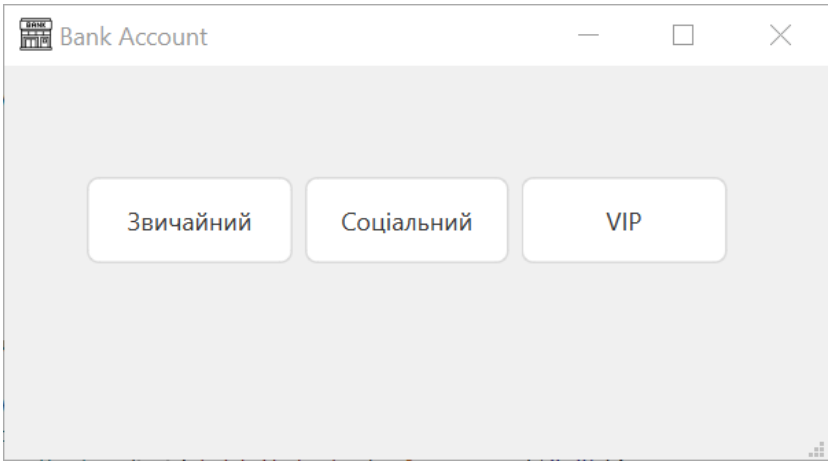

Рис 1. Початкове вікно, вибір типу банківського рахунку

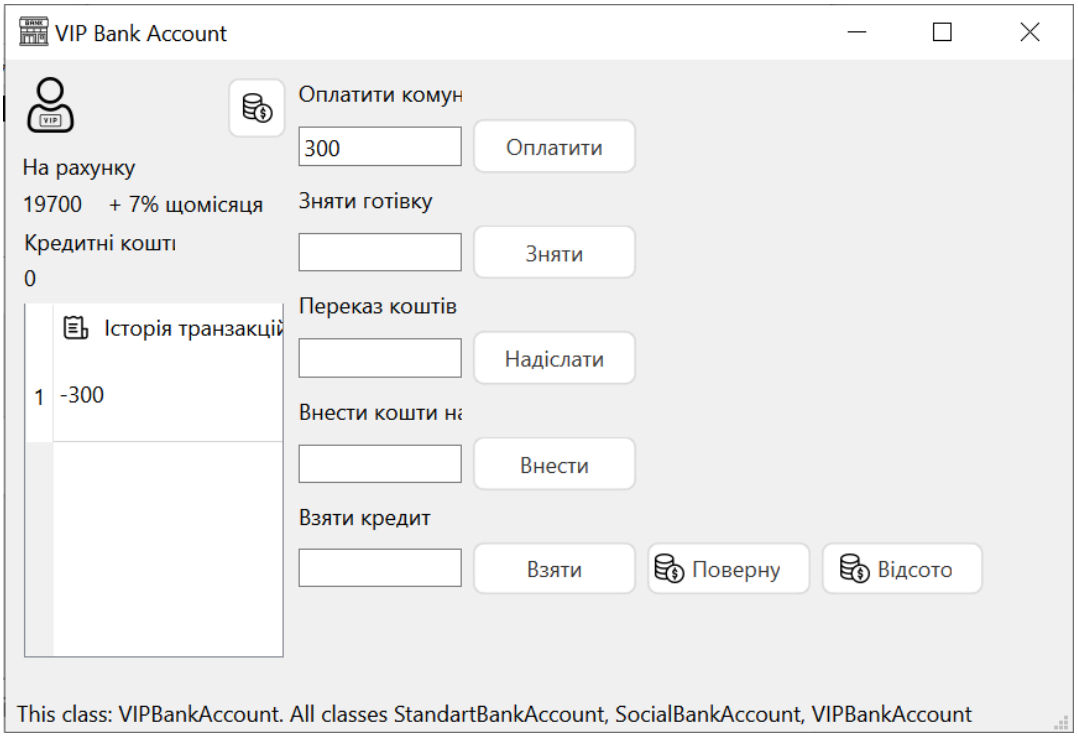

Рис 2. Вікно банківського віп рахунку

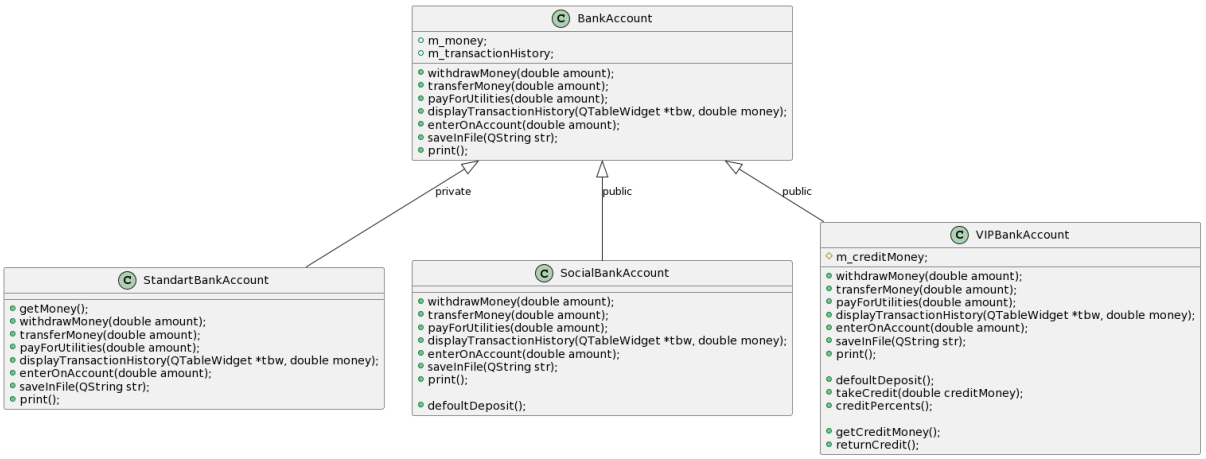

Рис 3.UML-діаграма

ВИСНОВКИ

Виконавши лабораторну роботу №8 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.