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 каф. ПЗ

Коротєєва Т.О.

Виконала: ст.гр. ПЗ-23

Кохман О.В.

Прийняла: доцент каф. ПЗ

Коротєєва Т.О.

«_____»_____2022р.

∑ ____ _____.

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.

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.

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 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, 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

Назва файлу: Standard.h

```
#ifndef STANDARD_H
#define STANDARD_H
#pragma once
```

```
#include <string>
using namespace std;
class Standard {
protected:
	string number;
	string password;
	double balance;
	double commission;
public:
	Standard();
	Standard(string number, string password);
	bool transferFunds(double sum);
	bool withdrawCash(double sum);
	void topUp(double sum);
	bool payUtilityBill(double sum);
	double getBalance() const;
	virtual string print() const;
};
#endif
```

Назва файлу: Standard.cpp

```
#include "Standard.h"
Standard::Standard() {
	number = "0000 0000 0000 0000";
	password = "0000";
	balance = 0;
	commission = 0.05;
}
Standard::Standard(string number, string password) {
	this->number = number;
	this->password = password;
	balance = 0;
	commission = 0.05;
}
bool Standard::transferFunds(double sum) {
	double commission = sum * this->commission;
	if (this->balance - (sum + commission) < 0) {
		return false;
	}
	else {
		this->balance -= (sum + commission);
		return true;
	}
}
bool Standard::withdrawCash(double sum) {
	double commission = sum * this->commission;
	if (this->balance - (sum+commission) < 0) {
		return false;
	}
	else {
		this->balance -= (sum + commission);
		return true;
	}
}
void Standard::topUp(double sum) {
	this->balance += sum;
}
bool Standard::payUtilityBill(double sum) {
	double commission = sum * this->commission;
	if ((this->balance - (sum + commission)) < 0) {
		return false;
	}
```

```
	else {
		this->balance -= (sum + commission);
		return true;
	}
}
double Standard::getBalance() const {
	return this->balance;
}
string Standard::print() const {
	return "Standard";
}
```

Назва файлу: Social.h

```
#ifndef SOCIAL_H
#define SOCIAL_H
#pragma once
#include "Standard.h"
class Social : public Standard {
protected:
	double bonusPercent;
private:
	string type;
public :
	Social();
	Social(string type);
	Social(string number, string password);
	Social(string number, string password, string type);
	virtual double getBonuses() const;
	void addBonuses(double sum);
	virtual string print() const override;
};
#endif
```

Назва файлу: Social.cpp

```
#include "Social.h"
Social::Social() {
	Standard();
	this->type = "benefit"; // student, pensioner, parent, orphan, benefit
	this->bonusPercent = 0.05;
	this->commission = 0;
}
Social::Social(string type) {
	Standard();
	this->type = type;
	this->bonusPercent = 0.05;
	this->commission = 0;
}
Social::Social(string number, string password) {
	Standard(number, password);
}
Social::Social(string number, string password, string type) {
	Standard(number, password);
	this->type = type;
	this->bonusPercent = 0.05;
	this->commission = 0;
}
double Social::getBonuses() const {
	if (balance >= 1000 && (int)balance % 1000 == 0) {
		return balance * bonusPercent;
	}
	else {
		return 0;
```

```
	}
}
void Social::addBonuses(double sum) {
	this->balance += sum;
}
string Social::print() const{
	return "Social";
}
```

Назва файлу: VIP.h

```
#ifndef VIP_H
#define VIP_H
#include "Social.h"
class VIP : protected Social {
private:
	double creditLimit;
	double creditPercent;
	double generalCredit;
public:
	VIP();
	VIP(string _number, string _password);
	double getGeneralCredit() const;
	void takeCredit(double sum);
	void repayCredit(double sum);
	using Social::getBalance;
	virtual double getBonuses() const;
	using Social::addBonuses;
	using Social::topUp;
	using Social::transferFunds;
	using Social::withdrawCash;
	using Social::payUtilityBill;
	virtual string print() const override;
};
#endif
```

Назва файлу: VIP.cpp

```
#include "VIP.h"
VIP::VIP() {
	Social();
	this->creditLimit = 50000;
	this->creditPercent = 20;
	this->generalCredit = 0;
	this->bonusPercent = 0.1;
};
VIP::VIP(string _number, string _password) {
	Social(_number, _password);
	this->creditLimit = 50000;
	this->creditPercent = 20;
	this->generalCredit = 0;
	this->bonusPercent = 0.1;
}
double VIP::getBonuses() const {
	if (balance >= 10000 && (int)balance % 1000 == 0) {
		return balance * bonusPercent;
	}
	else if (balance >= 1000 && (int)balance % 1000 == 0) {
		double bonusPercent2 = 0.05;
		return balance* bonusPercent2;
	}
	else {
		return 0;
	}
```

```
}
void VIP::takeCredit(double sum) {
	if (creditLimit - sum >= 0) {
		generalCredit += (sum * creditPercent / 100) + sum;
		creditLimit -= sum;
		balance += sum;
	}
	else {
		return;
	}
}
void VIP::repayCredit(double sum) {
	if (generalCredit - sum >= 0 && balance-sum >=0) {
		generalCredit -= sum;
		creditLimit += sum;
		balance -= sum;
	}
	else {
		return;
	}
}
string VIP::print() const {
	return "VIP";
}
double VIP::getGeneralCredit() const {
	return generalCredit;
}
```

Назва файлу: MyForm.h

```
#pragma once
#include "Standard.h"
#include "StringConvertion.h"
#include "Social.h"
#include "VIP.h"
#include <iostream>
#include <fstream>
namespace BankAccount2 {
	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;
	public ref class MyForm : public System::Windows::Forms::Form
	{
	public:
		MyForm(void)
		{
			InitializeComponent();
		}
	protected:
		~MyForm()
		{
			if (components)
			{
				delete components;
			}
		}
	private: System::Windows::Forms::Button^ button8;
	protected:
	private: System::Windows::Forms::Label^ label6;
	private: System::Windows::Forms::TextBox^ textBox11;
```

```
	private: System::Windows::Forms::TextBox^ textBox10;
	private: System::Windows::Forms::TextBox^ textBox9;
	private: System::Windows::Forms::TextBox^ textBox8;
	private: System::Windows::Forms::TextBox^ textBox6;
	private: System::Windows::Forms::TextBox^ textBox5;
	private: System::Windows::Forms::TextBox^ textBox4;
	private: System::Windows::Forms::TextBox^ textBox3;
	private: System::Windows::Forms::Label^ label5;
	private: System::Windows::Forms::Label^ label4;
	private: System::Windows::Forms::Button^ button7;
	private: System::Windows::Forms::Button^ button6;
	private: System::Windows::Forms::Button^ button5;
	private: System::Windows::Forms::Button^ button4;
	private: System::Windows::Forms::Button^ button3;
	private: System::Windows::Forms::Button^ button2;
	private: System::Windows::Forms::DataGridView^ transactionHistory;
	private: System::Windows::Forms::DataGridView^ Info;
	private: System::Windows::Forms::Button^ button1;
	private: System::Windows::Forms::Label^ label3;
	private: System::Windows::Forms::Label^ label2;
	private: System::Windows::Forms::Label^ label1;
	private: System::Windows::Forms::ComboBox^ comboBox1;
	private: System::Windows::Forms::TextBox^ textBox2;
	private: System::Windows::Forms::TextBox^ textBox1;
	private: System::Windows::Forms::ComboBox^ comboBox2;
	private: System::Windows::Forms::Label^ label7;
	private: System::Windows::Forms::Button^ button9;
	private: System::Windows::Forms::Button^ button10;
	private: System::Windows::Forms::TextBox^ textBox7;
	private: System::Windows::Forms::Button^ button11;
	private: System::Windows::Forms::Label^ label8;
	private:
		System::ComponentModel::Container ^components;
#pragma region Windows Form Designer generated code

#pragma endregion
		int type = 0;
		int typeSpec = 0;
		Standard* standard;
		Social* social;
		VIP* vip;
private: System::Void createButton(System::Object^ sender, System::EventArgs^ e)
{
	String^ tempType = comboBox1->Text;
	String^ tempSpec = comboBox2->Text;
	Info->RowCount = 1;
	Info->ColumnCount = 7;
	transactionHistory->ColumnCount = 5;
	Info->Columns[0]->HeaderText = "CardNumber";
	Info->Columns[1]->HeaderText = "Type";
	Info->Columns[2]->HeaderText = "Balance";
	Info->Columns[3]->HeaderText = "Commission";
	Info->Columns[4]->HeaderText = "BonusBalance";
	Info->Columns[5]->HeaderText = "CreditPercent";
	Info->Columns[6]->HeaderText = "Repayment";
	String^ tempNumber = textBox2->Text;
	Info->Rows[0]->Cells[0]->Value = tempNumber;
	if (tempSpec == "Benefit") {
		typeSpec = 1;
	}
	else if (tempSpec == "Student") {
		typeSpec = 2;
	}
```

```
	else if (tempSpec == "Pensioner") {
		typeSpec = 3;
	}
	else if (tempSpec == "Orphan") {
		typeSpec = 4;
	}
	else if (tempSpec == "Parent") {
		typeSpec = 5;
	}
	if (tempType == "Standard") {
		type = 1;
		Info->Rows[0]->Cells[1]->Value = "Standard";
		Info->Rows[0]->Cells[2]->Value = "0.0";
		Info->Rows[0]->Cells[3]->Value = "0.05%";
		Info->Rows[0]->Cells[4]->Value = "unavailable";
		Info->Rows[0]->Cells[5]->Value = "unavailable";
		Info->Rows[0]->Cells[6]->Value = "unavailable";
	}
	else if (tempType == "Social") {
		type = 2;
		Info->Rows[0]->Cells[1]->Value = "Social: " + tempSpec;
		Info->Rows[0]->Cells[2]->Value = "0.0";
		Info->Rows[0]->Cells[3]->Value = "0.0%";
		Info->Rows[0]->Cells[4]->Value = "0.0";
		Info->Rows[0]->Cells[5]->Value = "unavailable";
		Info->Rows[0]->Cells[6]->Value = "unavailable";
	}
	else if (tempType == "VIP") {
		type = 3;
		Info->Rows[0]->Cells[1]->Value = "VIP";
		Info->Rows[0]->Cells[2]->Value = "0.0";
		Info->Rows[0]->Cells[3]->Value = "0.0%";
		Info->Rows[0]->Cells[4]->Value = "0.0";
		Info->Rows[0]->Cells[5]->Value = "20%";
		Info->Rows[0]->Cells[6]->Value = "0.0";
	}

}

private: System::Void buttonsClick(System::Object^ sender, System::EventArgs^ e)
{
	std::ofstream file("output.txt", ios::app);
	System::Windows::Forms::Button^ button = (Button^)sender;
	System::String^ textOfButtons = button->Text;
	String^ tempPassword = textBox1->Text;
	String^ tempNumber = textBox2->Text;
	String^ tempPasswordAgain = textBox11->Text;
	String^ tempSpec = comboBox2->Text;
	string spec = toStandardString(tempSpec);
	string password = toStandardString(tempPassword);
	string number = toStandardString(tempNumber);
	string passwordAgain = toStandardString(tempPasswordAgain);
	if (textOfButtons == "Confirm") {
		if (type == 1) {
			button2->Enabled = true;
			button3->Enabled = true;
			button4->Enabled = true;
			button5->Enabled = true;
			standard = new Standard(number, password);
		}
		else if (type == 2) {
			button2->Enabled = true;
			button3->Enabled = true;
```

```
                        button4->Enabled = true;
                        button5->Enabled = true;
                        button6->Enabled = true;
                        button9->Enabled = true;
                        social = new Social(number, password, spec);
                }
                else if (type == 3) {
                        button2->Enabled = true;
                        button3->Enabled = true;
                        button4->Enabled = true;
                        button5->Enabled = true;
                        button6->Enabled = true;
                        button7->Enabled = true;
                        button9->Enabled = true;
                        button10->Enabled = true;
                        vip = new VIP(number, password);
                }
                button8->Enabled = false;
                button11->Enabled = true;
        }
        else if (textOfButtons == "topUp") {
                double sum = System::Convert::ToDouble(textBox3->Text);
                double currentBalance = 0;
                if (type == 1) {
                        standard->topUp(sum);
                        Info->Rows[0]->Cells[2]->Value =
System::Convert::ToString(standard->getBalance());
                        currentBalance = standard->getBalance();
                }
                else if (type == 2) {
                        social->topUp(sum);
                        Info->Rows[0]->Cells[2]->Value =
System::Convert::ToString(social->getBalance());
                        currentBalance = social->getBalance();
                }
                else if(type == 3) {
                        vip->topUp(sum);
                        Info->Rows[0]->Cells[2]->Value =
System::Convert::ToString(vip->getBalance());
                        currentBalance = vip->getBalance();
                }
                transactionHistory->Rows->Insert(0, "top up", tempNumber, sum,
"added", currentBalance);
                file << "top up" << " " << toStandardString(tempNumber) << " " <<
to_string(sum) << " " << "added " << to_string(currentBalance) << "\n";
        }
        else if (textOfButtons == "withdrawCash") {
                double sum = System::Convert::ToDouble(textBox5->Text);
                double currentBalance;
                if (type == 1) {
                        if (standard->withdrawCash(sum)) {
                                Info->Rows[0]->Cells[2]->Value =
System::Convert::ToString(standard->getBalance());
                                currentBalance = standard->getBalance();
                        }
                }
                else if (type == 2) {
                        if (social->withdrawCash(sum)) {
                                Info->Rows[0]->Cells[2]->Value =
System::Convert::ToString(social->getBalance());
                                currentBalance = social->getBalance();
                        }
                }
```

```
            else if (type == 3) {
                  if (vip->withdrawCash(sum)) {
                        Info->Rows[0]->Cells[2]->Value =
System::Convert::ToString(vip->getBalance());
                  }
                  currentBalance = vip->getBalance();
            }
            transactionHistory->Rows->Insert(0, "withdraw cash", tempNumber,
sum, "substracted", currentBalance);
            file << "withdraw cash" << " " << toStandardString(tempNumber) << "
" << to_string(sum) << " " << "substracted " << to_string(currentBalance) <<
"\n";
      }
      else if (textOfButtons == "transferFunds") {
            String^ tempAnotherAccount = textBox6->Text;
            string anotherAccount = toStandardString(tempAnotherAccount);
            double sum = System::Convert::ToDouble(textBox9->Text);
            double currentBalance;
            if (type == 1) {
                  if (standard->transferFunds(sum)) {
                        Info->Rows[0]->Cells[2]->Value =
System::Convert::ToString(standard->getBalance());
                        currentBalance = standard->getBalance();
                  }
            }
            else if (type == 2) {
                  if (social->transferFunds(sum)) {
                        Info->Rows[0]->Cells[2]->Value =
System::Convert::ToString(social->getBalance());
                        currentBalance = social->getBalance();
                  }
            }
            else if (type == 3) {
                  if (vip->transferFunds(sum)) {
                        Info->Rows[0]->Cells[2]->Value =
System::Convert::ToString(vip->getBalance());
                        currentBalance = vip->getBalance();
                  }
            }
            transactionHistory->Rows->Insert(0, "transfer funds",
tempAnotherAccount, sum, "substracted", currentBalance );
            file << "transfer funds" << " " <<
toStandardString(tempAnotherAccount) << " " << to_string(sum) << " " <<
"substracted " << to_string(currentBalance) << "\n";
      }
      else if (textOfButtons == "getBonuses") {
            if (type == 2) {
                  Info->Rows[0]->Cells[4]->Value =
System::Convert::ToString(social->getBonuses());
            }
            else if (type == 3) {
                  Info->Rows[0]->Cells[4]->Value =
System::Convert::ToString(vip->getBonuses());
            }
      }
      else if (textOfButtons == "transferBonuses") {
            if (type == 2) {
                  double bonuses = social->getBonuses();
                  social->addBonuses(bonuses);
                  Info->Rows[0]->Cells[2]->Value =
System::Convert::ToString(social->getBalance());
                  Info->Rows[0]->Cells[4]->Value = "0.0";
```

```
                  transactionHistory->Rows->Insert(0, "transfer bonuses",
tempNumber, social->getBonuses(), "added", social->getBonuses());
                  file << "transfer bonuses" << " " <<
toStandardString(tempNumber) << " " << to_string(bonuses) << " " << "added " <<
to_string(social->getBalance()) << "\n";
            }
            else if (type == 3) {
                  vip->addBonuses(vip->getBonuses());
                  Info->Rows[0]->Cells[2]->Value =
System::Convert::ToString(vip->getBalance());
                  Info->Rows[0]->Cells[4]->Value = "0.0";
                  transactionHistory->Rows->Insert(0, "transfer bonuses",
tempNumber, vip->getBonuses(), "added", vip->getBonuses());
                  file << "transfer bonuses" << " " <<
toStandardString(tempNumber) << " " << to_string(vip->getBonuses()) << " " <<
"added " << to_string(vip->getBalance()) << "\n";
            }
      }
      else if (textOfButtons == "takeACredit") {
            double sum = System::Convert::ToDouble(textBox8->Text);
            vip->takeCredit(sum);
            Info->Rows[0]->Cells[2]->Value = System::Convert::ToString(vip-
>getBalance());
            Info->Rows[0]->Cells[6]->Value = System::Convert::ToString(vip-
>getGeneralCredit());
            transactionHistory->Rows->Insert(0, "take a credit", tempNumber,
sum, "added", vip->getBalance());
            file << "take a credit" << " " << toStandardString(tempNumber) << "
" << to_string(sum) << " " << "added " << to_string(vip->getBalance()) << "\n";
      }
      else if (textOfButtons == "repayACredit") {
          double sum = System::Convert::ToDouble(textBox7->Text);
            vip->repayCredit(sum);
            Info->Rows[0]->Cells[2]->Value = System::Convert::ToString(vip-
>getBalance());
            Info->Rows[0]->Cells[6]->Value = System::Convert::ToString(vip-
>getGeneralCredit());
            transactionHistory->Rows->Insert(0, "repay a credit", tempNumber,
sum, "substracted", vip->getBalance());
            file << "repay a credit" << " " << toStandardString(tempNumber) << "
" << to_string(sum) << " " << "substacted " << to_string(vip->getBalance()) <<
"\n";
      }
      else if (textOfButtons == "payTheUtilityBill") {
            String^ tempBill = textBox4->Text;
            string bill = toStandardString(tempBill);
            double sum = System::Convert::ToDouble(textBox10->Text);
            double currentBalance;
            if (type == 1) {
                  if (standard->payUtilityBill(sum)) {
                        Info->Rows[0]->Cells[2]->Value =
System::Convert::ToString(standard->getBalance());
                        currentBalance = standard->getBalance();
                  }
            }
            else if (type == 2) {
                  if (social->payUtilityBill(sum)) {
                        Info->Rows[0]->Cells[2]->Value =
System::Convert::ToString(social->getBalance());
                        currentBalance = social->getBalance();
                  }
            }
            else if (type == 3) {
```

```
                    if (vip->payUtilityBill(sum)) {
                          Info->Rows[0]->Cells[2]->Value =
System::Convert::ToString(vip->getBalance());
                          currentBalance = vip->getBalance();
                    }
              }
              transactionHistory->Rows->Insert(0, "pay the utility bill",
tempBill, sum, "substracted", currentBalance);
              file << "pay the utility bill" << " " << toStandardString(tempBill)
<<" " << to_string(sum) << " " << "substacted " << to_string(currentBalance) <<
"\n";
       }
}
private: System::Void button11_Click(System::Object^ sender, System::EventArgs^
e) {
       String^ str;
       if (type == 1) {
           str = gcnew String(standard->print().c_str());
       }
       else if (type == 2) {
              str = gcnew String(social->print().c_str());
       }
       else if (type == 3) {
              str = gcnew String(vip->print().c_str());
       }
       label8->Text = str;
}
};
}
```

Назва файлу: MyForm.cpp

```
#include "MyForm.h"
using namespace BankAccount2;
int main() {
       Application::EnableVisualStyles();
       Application::SetCompatibleTextRenderingDefault(false);
       Application::Run(gcnew MyForm());
       return 0;
}
```

Протокол роботи

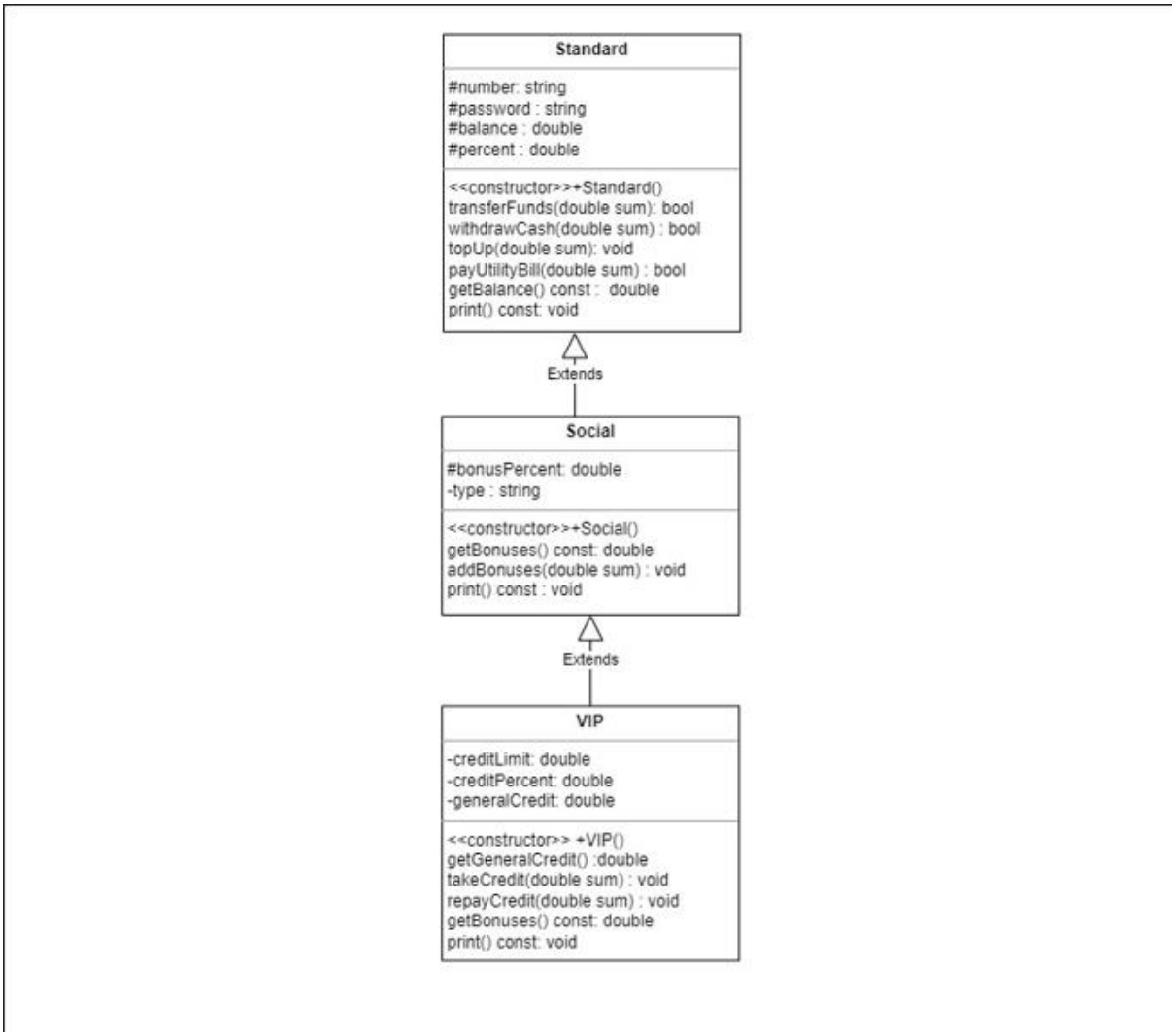

Рис. 1 UML-діаграма розробленої ієрархії класів.

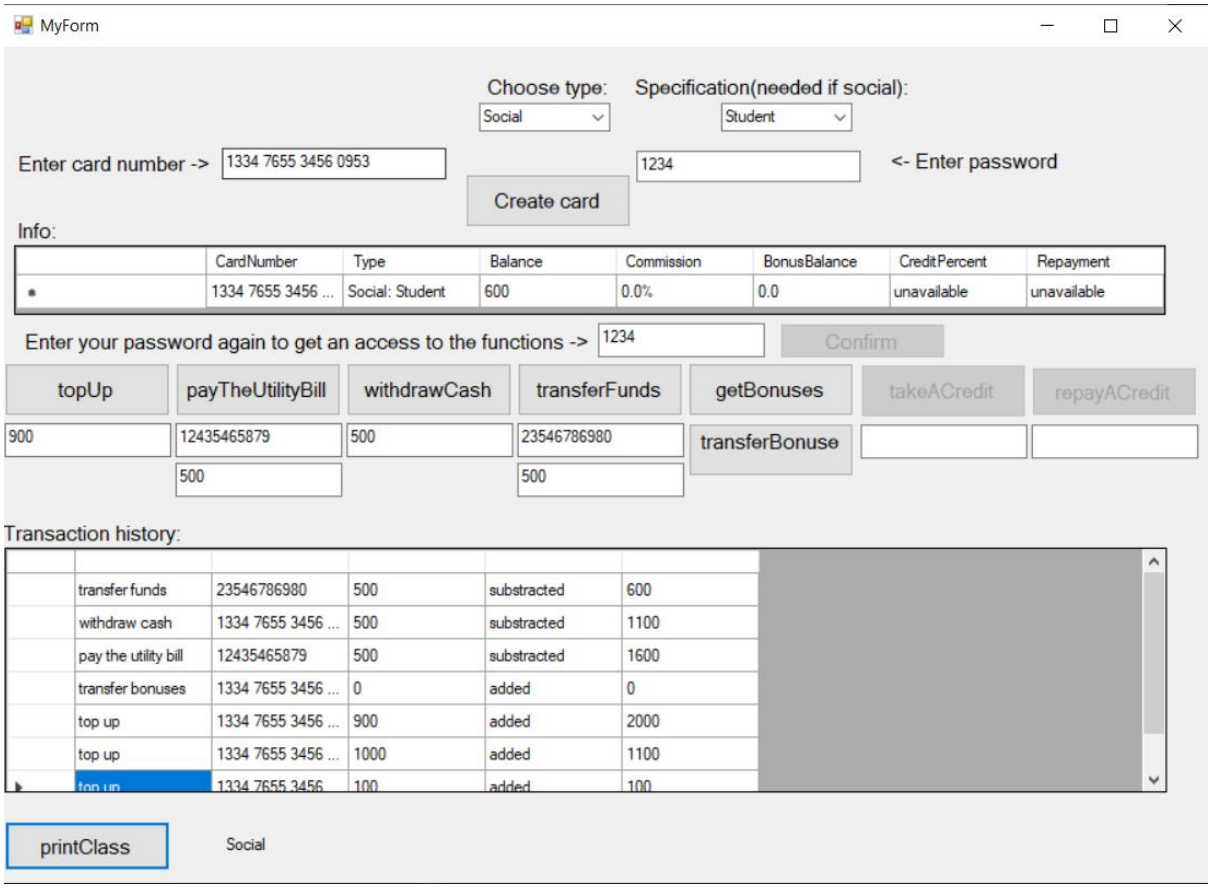

Рис. 2 Результат виконання програми.

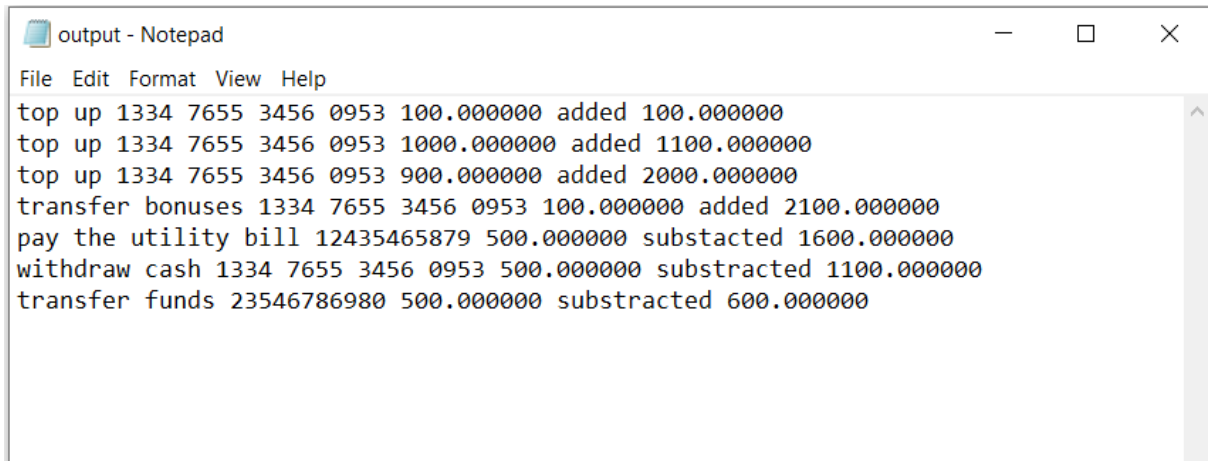

Рис. 3 Дані, записані через програму у файл.

Висновок

На цій лабораторній роботі я дізналась про наслідування та поліморфізм в об’єктно-орієнтованому програмуванні , а також реалізувала свою ієрархію класів і продемонструвала результати роботи програми у віконному додатку за допомогою Visual Studio 2022.