Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

Кафедра кібербезпеки та програмного забезпечення

ЗВІТ

ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ №4

з дисципліни «Об’єктно-орієнтоване програмування»

**Тема:** «Поліморфізм. Обробка винятків»

|  |
| --- |
| Виконав: ст. гр. КН-23 |
| Гребенюк Д.О. |
| Перевірив: асистент  Козірова Н.Л. |
|  |

Кропивницький

2024

**Мета:** Ознайомитись з поняттям поліморфізму у мові C++ та навчитись використовувати віртуальні функції для досягнення поліморфізму. Також вивчити принципи обробки винятків у мові C++.

**Варіант 4**

**Завдання 1**

Напишіть гру «Піймай муху», на формі знаходиться муха, яка має тікати від курсора, також на формі має знаходитися пастка, коли муха попадеться в пастку гра закінчується. В реалізації програми має бути похідний клас, що унаслідується від базового класу «QMainWindow», в похідному класі, перевизначить функцію «event»

**Завдання 2**

1. Розробіть систему керування банком, яка включатиме обробку різних типів банківських рахунків, таких як звичайний рахунок та рахунок з відсотковою ставкою. Кожен тип рахунку має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен банківський рахунок має наступні характеристики:

• Номер рахунку

• Власник рахунку

• Баланс

Кожен тип рахунку має власні додаткові характеристики:

Звичайний рахунок:

• Мінімальний дозволений баланс

Рахунок з відсотковою ставкою:

• Відсоткова ставка

2. Створіть базовий абстрактний клас BankAccount з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи RegularAccount та InterestAccount, які успадковуються від класу BankAccount. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо некоректний номер рахунку або негативна відсоткова ставка. Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

**Завдання 1**

*Лістинг mainwindow.h*

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#define FLY\_MOVE\_DIST\_CONDITION 70

QT\_BEGIN\_NAMESPACE

namespace Ui {

class MainWindow;

}

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private:

Ui::MainWindow \*ui;

QPoint flyInitialPos;

bool event(QEvent \* event) override;

int generateRandomXY();

double calculateDistance(QPoint &firstPoint, QPoint &secondPoint);

};

#endif // MAINWINDOW\_H

*Лістинг mainwindow.cpp*

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <cmath>

#include <QRandomGenerator>

#include <QMessageBox>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

QPixmap flyPixmap(":/assets/fly.png");

ui->flyObject->setIcon(QIcon(flyPixmap));

ui->flyObject->setIconSize(QSize(64, 64));

flyInitialPos = ui->flyObject->pos();

QPixmap trapPlantPixmap(":/assets/plant\_trap.png");

ui->trapPlantObject->setIcon(QIcon(trapPlantPixmap));

ui->trapPlantObject->setIconSize(QSize(256, 256));

ui->trapPlantObject->stackUnder(ui->flyObject);

this->setMouseTracking(true);

ui->centralwidget->setMouseTracking(true);

ui->flyObject->setMouseTracking(true);

}

MainWindow::~MainWindow()

{

delete ui;

}

bool MainWindow::event(QEvent \*event)

{

if (event->type() == QEvent::MouseMove) {

QPoint mousePos = ui->centralwidget->mapFromGlobal(QCursor::pos());

QPoint flyPos = ui->flyObject->pos();

QPoint trapPlantPos = ui->trapPlantObject->pos();

double mouseDistance = calculateDistance(flyPos, mousePos);

if (mouseDistance <= FLY\_MOVE\_DIST\_CONDITION) {

QPoint newFlyPos(0, 0);

while (true) {

int randomX = generateRandomXY();

int randomY = generateRandomXY();

newFlyPos.setX(flyPos.x() + randomX);

newFlyPos.setY(flyPos.y() + randomY);

int windowMoveLimitForX = this->size().width() - ui->flyObject->iconSize().width();

if (newFlyPos.x() < 0 || newFlyPos.x() > windowMoveLimitForX)

continue;

int windowMoveLimitForY = this->size().height() - ui->flyObject->iconSize().height();

if (newFlyPos.y() < 0 || newFlyPos.y() > windowMoveLimitForY)

continue;

break;

}

ui->flyObject->move(newFlyPos);

}

if (ui->trapPlantObject->geometry().contains(flyPos)) {

QMessageBox msgBox;

msgBox.setIcon(QMessageBox::Information);

msgBox.setText("Муха спіймана, кінець гри :)");

msgBox.exec();

ui->flyObject->move(flyInitialPos);

}

}

return QMainWindow::event(event);

}

int MainWindow::generateRandomXY() {

return QRandomGenerator::global()->bounded(-FLY\_MOVE\_DIST\_CONDITION, FLY\_MOVE\_DIST\_CONDITION);

}

double MainWindow::calculateDistance(QPoint &firstPoint, QPoint &secondPoint) {

return sqrt(pow(secondPoint.x() - firstPoint.x(), 2) + pow(secondPoint.y() - firstPoint.y(), 2));

}

*Лістинг mainwindow.ui*

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>MainWindow</class>

<widget class="QMainWindow" name="MainWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>800</width>

<height>600</height>

</rect>

</property>

<property name="windowTitle">

<string>Лабораторна робота №4</string>

</property>

<widget class="QWidget" name="centralwidget">

<widget class="QPushButton" name="flyObject">

<property name="geometry">

<rect>

<x>10</x>

<y>240</y>

<width>64</width>

<height>64</height>

</rect>

</property>

<property name="styleSheet">

<string notr="true">background-color: rgba(255, 255, 255, 0);

border: none;</string>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QPushButton" name="trapPlantObject">

<property name="geometry">

<rect>

<x>530</x>

<y>150</y>

<width>256</width>

<height>256</height>

</rect>

</property>

<property name="styleSheet">

<string notr="true">background-color: rgba(255, 255, 255, 0);

border: none;</string>

</property>

<property name="text">

<string/>

</property>

</widget>

</widget>

<widget class="QMenuBar" name="menubar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>800</width>

<height>18</height>

</rect>

</property>

</widget>

<widget class="QStatusBar" name="statusbar"/>

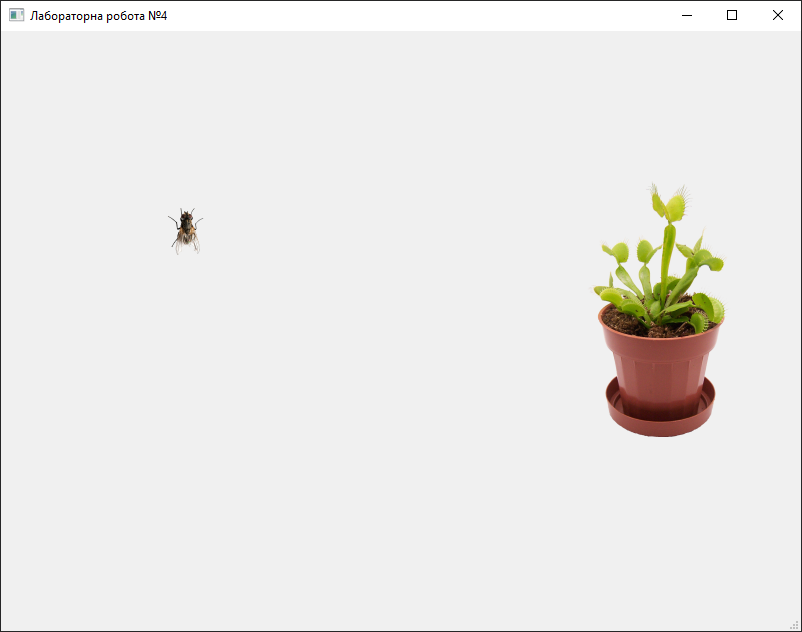
</widget>

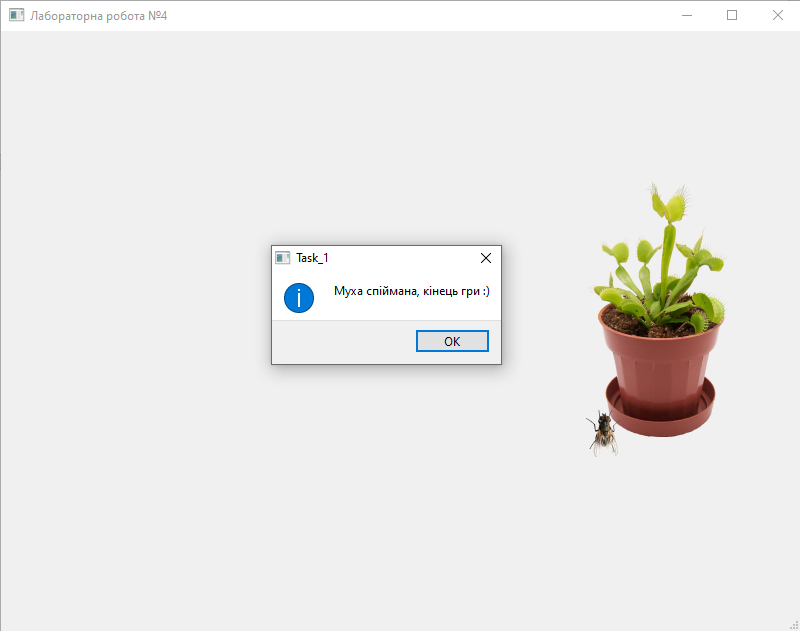
<resources/>

<connections/>

</ui>

*Результат виконання*





**Завдання 2**

*Лістинг main.cpp*

#include <iostream>

#include <vector>

#include "bankaccount.h"

#include "regularaccount.h"

#include "interestaccount.h"

void showMenu() {

std::cout << "\n1. Create a new regular account\n";

std::cout << "2. Create a new interest account\n";

std::cout << "3. Deposit to account\n";

std::cout << "4. Withdraw from account\n";

std::cout << "5. Switch account\n";

std::cout << "6. Show balance\n";

std::cout << "7. Exit\n";

std::cout << "Choose an option: ";

}

int main() {

std::vector<BankAccount\*> accounts;

int currentAccount = -1;

while (true) {

showMenu();

int choice;

std::cin >> choice;

if (choice == 1) {

int number;

std::string owner;

long int balance, minimumBalance;

std::cout << "Enter account number: ";

std::cin >> number;

std::cout << "Enter owner name: ";

std::cin >> owner;

std::cout << "Enter starting balance: ";

std::cin >> balance;

std::cout << "Enter minimum allowed balance: ";

std::cin >> minimumBalance;

accounts.push\_back(new RegularAccount(number, owner, balance, minimumBalance));

currentAccount = accounts.size() - 1;

} else if (choice == 2) {

int number;

std::string owner;

long int balance;

int interestRate;

std::cout << "Enter account number: ";

std::cin >> number;

std::cout << "Enter owner name: ";

std::cin >> owner;

std::cout << "Enter starting balance: ";

std::cin >> balance;

std::cout << "Enter interest rate: ";

std::cin >> interestRate;

accounts.push\_back(new InterestAccount(number, owner, balance, interestRate));

currentAccount = accounts.size() - 1;

} else if (choice == 3) {

if (currentAccount < 0) {

std::cout << "No account selected.\n";

continue;

}

double amount;

std::cout << "Enter amount to deposit: ";

std::cin >> amount;

accounts[currentAccount]->deposit(amount);

std::cout << "Deposit successful.\n";

} else if (choice == 4) {

if (currentAccount < 0) {

std::cout << "No account selected.\n";

continue;

}

double amount;

std::cout << "Enter amount to withdraw: ";

std::cin >> amount;

accounts[currentAccount]->withdraw(amount);

std::cout << "Withdrawal attempt completed.\n";

} else if (choice == 5) {

int accountIndex;

std::cout << "Enter account index to switch (0 - " << accounts.size() - 1 << "): ";

std::cin >> accountIndex;

if (accountIndex >= 0 && accountIndex < accounts.size()) {

currentAccount = accountIndex;

std::cout << "Switched to account " << currentAccount << ".\n";

} else {

std::cout << "Invalid account index.\n";

}

} else if (choice == 6) {

if (currentAccount < 0) {

std::cout << "No account selected.\n";

continue;

}

std::cout << "Current balance: " << accounts[currentAccount]->getBalance() << "\n";

} else if (choice == 7) {

break;

} else {

std::cout << "Invalid choice.\n";

}

}

for (BankAccount\* account : accounts) {

delete account;

}

return 0;

}

*Лістинг bankaccount.cpp*

#include "bankaccount.h"

#include <iostream>

BankAccount::BankAccount(int number, const std::string &owner, long int balance)

{

setNumber(number);

setOwner(owner);

setBalance(balance);

}

int BankAccount::getNumber() const

{

return number;

}

void BankAccount::setNumber(int newNumber)

{

if (newNumber <= 0) {

std::cout << "The account number cannot be less than or equal to zero." << std::endl;

return;

}

number = newNumber;

}

std::string BankAccount::getOwner() const

{

return owner;

}

void BankAccount::setOwner(const std::string &newOwner)

{

if (newOwner.empty()) {

std::cout << "You have not entered the name of the account holder." << std::endl;

return;

}

owner = newOwner;

}

long int BankAccount::getBalance() const

{

return balance;

}

void BankAccount::setBalance(long int newBalance)

{

balance = newBalance;

}

*Лістинг bankaccount.h*

#ifndef BANKACCOUNT\_H

#define BANKACCOUNT\_H

#include <string>

class BankAccount

{

public:

BankAccount(int number, const std::string &owner, long int balance);

int getNumber() const;

void setNumber(int newNumber);

std::string getOwner() const;

void setOwner(const std::string &newOwner);

long int getBalance() const;

void setBalance(long int newBalance);

virtual void withdraw(double amount) = 0;

virtual void deposit(double amount) = 0;

virtual double calculateInterest() const = 0;

protected:

int number;

std::string owner;

long int balance;

};

#endif // BANKACCOUNT\_H

*Лістинг interestaccount.cpp*

#include "interestaccount.h"

#include <iostream>

InterestAccount::InterestAccount(int number, const std::string &owner, long int balance, int interestRate) : BankAccount(number, owner, balance)

{

setInterestRate(interestRate);

}

int InterestAccount::getInterestRate() const

{

return interestRate;

}

void InterestAccount::setInterestRate(int newInterestRate)

{

if (newInterestRate <= 0) {

std::cout << "The interest rate cannot be equal to or less than zero." << std::endl;

return;

}

interestRate = newInterestRate;

}

void InterestAccount::withdraw(double amount) {

if (amount > balance) {

std::cout << "There are not enough money in your account." << std::endl;

return;

}

balance -= amount;

}

void InterestAccount::deposit(double amount) {

if (amount <= 0) {

std::cout << "The deposit amount cannot be less than or equal to zero." << std::endl;

return;

}

balance += amount;

balance += calculateInterest();

}

double InterestAccount::calculateInterest() const {

return balance \* interestRate / 100;

}

*Лістинг interestaccount.h*

#ifndef INTERESTACCOUNT\_H

#define INTERESTACCOUNT\_H

#include "bankaccount.h"

class InterestAccount : public BankAccount

{

public:

InterestAccount(int number, const std::string &owner, long int balance, int interestRate);

int getInterestRate() const;

void setInterestRate(int newInterestRate);

void withdraw(double amount) override;

void deposit(double amount) override;

double calculateInterest() const override;

private:

int interestRate;

};

#endif // INTERESTACCOUNT\_H

*Лістинг regularaccount.cpp*

#include "regularaccount.h"

#include <iostream>

RegularAccount::RegularAccount(int number, const std::string &owner, long int balance, long int minimumAllowedBalance) : BankAccount(number, owner, balance)

{

setMinimumAllowedBalance(minimumAllowedBalance);

}

long int RegularAccount::getMinimumAllowedBalance() const

{

return minimumAllowedBalance;

}

void RegularAccount::setMinimumAllowedBalance(long int newMinimumAllowedBalance)

{

if (newMinimumAllowedBalance < 0) {

std::cout << "The minimum allowed balance cannot be less than zero." << std::endl;

return;

}

minimumAllowedBalance = newMinimumAllowedBalance;

}

void RegularAccount::withdraw(double amount) {

if (amount > balance) {

std::cout << "There are not enough money in your account." << std::endl;

return;

}

if (balance - amount < minimumAllowedBalance) {

std::cout << "Balance cannot be less than the minimum allowed balance." << std::endl;

return;

}

balance -= amount;

}

void RegularAccount::deposit(double amount) {

if (amount <= 0) {

std::cout << "The deposit amount cannot be less than or equal to zero." << std::endl;

return;

}

balance += amount;

}

double RegularAccount::calculateInterest() const {

return 0.0;

}

*Лістинг regularaccount.h*

#ifndef REGULARACCOUNT\_H

#define REGULARACCOUNT\_H

#include "bankaccount.h"

class RegularAccount : public BankAccount

{

public:

RegularAccount(int number, const std::string &owner, long int balance, long int minimumAllowedBalance);

long int getMinimumAllowedBalance() const;

void setMinimumAllowedBalance(long int newMinimumAllowedBalance);

void withdraw(double amount) override;

void deposit(double amount) override;

double calculateInterest() const override;

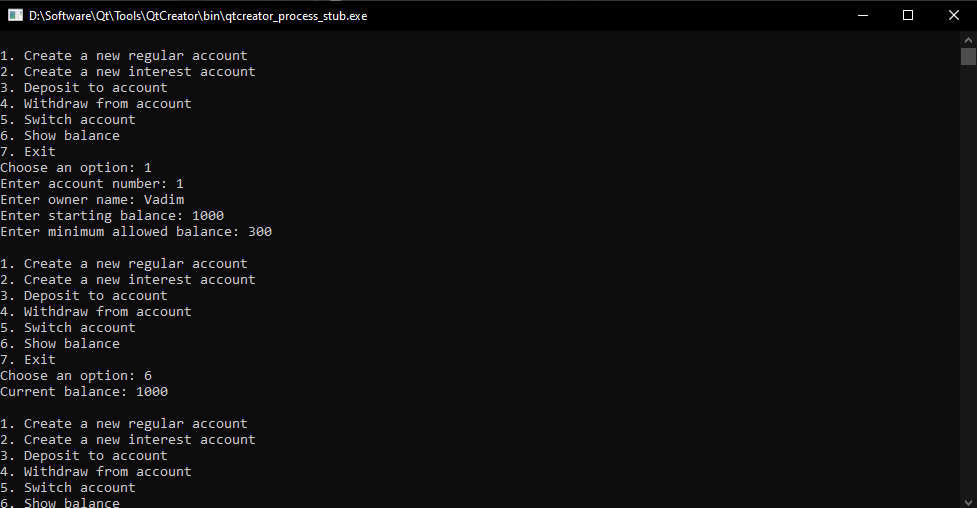
private:

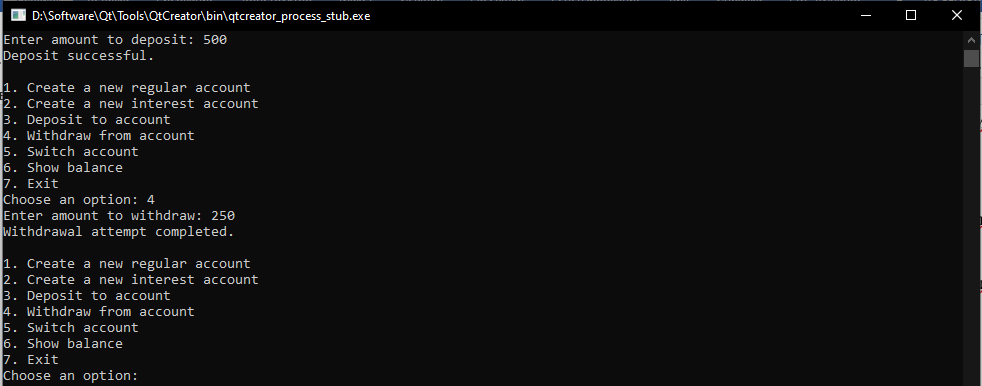
long int minimumAllowedBalance;

};

#endif // REGULARACCOUNT\_H

*Результат виконання*





**Висновок:**

Виконуючи лабораторну роботу, я ознайомився з концепцією поліморфізму в мові програмування C++ та засвоїв використання віртуальних функцій для його реалізації. Крім того, я навчився ефективно обробляти винятки для керування помилками в коді. Під час виконання завдань я реалізував гру «Піймай муху», яка демонструє наслідування і перевизначення функцій, а також розробив систему управління банківськими рахунками, що дозволила застосувати поліморфізм для обробки різних типів рахунків.