Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

Кафедра кібербезпеки та програмного забезпечення

Дисципліна: Об’єктно-орієнтоване програмування

**Лабораторна робота №4**

**Тема:** **«**ПОЛІМОРФІЗМ. ОБРОБКА ВИНЯТКІВ**»**

|  |
| --- |
| Виконав: ст. гр. КI-23 |
| Чепіль В. О. |
| Перевірив викладач:  Козірова Н. Л. |
|  |

Кропивницький 2024

**Тема:** Поліморфізм. Обробка винятків

**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:**

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

1. 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.
2. 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.
3. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо некоректний номер рахунку або негативна відсоткова ставка.

**Лістинг завдання 1(Task\_1):**

**Лістинг mainwindow.h:**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QGraphicsScene>

#include <QMouseEvent>

#include <QPixmap>

#include <QLabel>

#include <QTimer>

#include <QMessageBox>

#include <QEvent>

QT\_BEGIN\_NAMESPACE

namespace Ui {

class MainWindow;

}

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

protected:

void mousePressEvent(QMouseEvent \*event) override;

void mouseReleaseEvent(QMouseEvent \*event) override;

bool event(QEvent \*e);

private:

Ui::MainWindow \*ui;

QGraphicsScene \*scene;

QCursor cursor1;

QCursor cursor2;

bool isPressed;

QLabel \*trap;

QLabel \*fly;

void moveFly();

void checkCollision();

};

#endif // MAINWINDOW\_H

**Лістинг mainwindow.cpp:**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow), isPressed(false)

{

ui->setupUi(this);

scene = new QGraphicsScene(this);

ui->graphicsView->setScene(scene);

QPixmap cursorImage1 = QPixmap(":pictures/flySwatter1.png");

QPixmap cursorImage2 = QPixmap(":pictures/flySwatter2.png");

cursor1 = QCursor(cursorImage1);

cursor2 = QCursor(cursorImage2);

setCursor(cursor1);

trap = new QLabel(this);

QPixmap trapImage = QPixmap(":/pictures/trap2.png");

trapImage = trapImage.scaled(64, 64, Qt::KeepAspectRatio ,Qt::SmoothTransformation);

trap->setPixmap(trapImage);

trap->setGeometry(400, 300, 64, 64);

fly = new QLabel(this);

QPixmap flyImage = QPixmap(":/pictures/fly.png");

flyImage = flyImage.scaled(24, 24, Qt::KeepAspectRatio ,Qt::SmoothTransformation);

fly->setPixmap(flyImage);

fly->setGeometry(128, 128, 24, 24);

QTimer \*timer = new QTimer(this);

connect(timer, &QTimer::timeout, this, &MainWindow::moveFly);

timer->start(100);

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::mousePressEvent(QMouseEvent \*event)

{

if (event->button() == Qt::LeftButton && !isPressed)

{

setCursor(cursor2);

isPressed = true;

grabMouse();

}

}

void MainWindow::mouseReleaseEvent(QMouseEvent \*event)

{

if (event->button() == Qt::LeftButton && isPressed)

{

setCursor(cursor1);

isPressed = false;

releaseMouse();

}

}

bool MainWindow::MainWindow::event(QEvent \*e)

{

if (e->type() == QEvent::MouseMove)

{

moveFly();

}

return QMainWindow::event(e);

}

void MainWindow::moveFly()

{

QPoint cursorPos = mapFromGlobal(QCursor::pos());

int mouseX = cursorPos.x();

int mouseY = cursorPos.y();

int flyX = fly->x();

int flyY = fly->y();

int distanceX = mouseX - flyX;

int distanceY = mouseY - flyY;

if (distanceX < 100 && distanceY < 100)

{

flyX += (distanceX < 0) ? 20 : -20;

flyY += (distanceY < 0) ? 20 : -20;

}

flyX = qBound(0, flyX, width() - fly->width());

flyY = qBound(0, flyY, height() - fly->height());

fly->move(flyX, flyY);

checkCollision();

}

void MainWindow::checkCollision()

{

if (fly->geometry().intersects(trap->geometry()))

{

QMessageBox::information(this, "Game over", "You caught a fly!");

qApp->quit();

}

}

**Лістинг завдання 2(task\_2):**

**Лістинг** **bankaccount.h:**

#ifndef BANKACCOUNT\_H

#define BANKACCOUNT\_H

#include <QString>

using namespace std;

class BankAccount

{

public:

BankAccount();

BankAccount(const QString& accountNum, const QString& accOwner, double accBalance);

virtual ~BankAccount();

// Віртуальні функції

virtual void deposit(double amount) = 0;

virtual void withdraw(double amount) = 0;

virtual void displayAccountDetails() const = 0;

// Гетери та сетери

QString getAccountNumber();

QString getOwner();

double getBalance() ;

void setAccountNumber(QString& accountNum);

void setOwner(QString& accOwner);

void setBalance(double accBalance);

protected:

QString accountNumber;

QString accountHolder;

QString owner;

double balance;

};

#endif // BANKACCOUNT\_H

**Лістинг** **bankaccount.cpp:**

#include "bankaccount.h"

BankAccount::BankAccount() : accountNumber("Unknown"), accountHolder("Unknown"), balance(0.0) {}

BankAccount::BankAccount(const QString &accountNum, const QString &accOwner, double accBalance)

: accountNumber(accountNum), owner(accOwner), balance(accBalance)

{

if (balance < 0) {

throw invalid\_argument("Initial balance cannot be negative");

}

}

BankAccount::~BankAccount() {}

QString BankAccount::getAccountNumber()

{

return accountNumber;

}

QString BankAccount::getOwner()

{

return owner;

}

double BankAccount::getBalance()

{

return balance;

}

void BankAccount::setAccountNumber(QString &accountNum)

{

accountNumber = accountNum;

}

void BankAccount::setOwner(QString &accOwner)

{

owner = accOwner;

}

void BankAccount::setBalance(double accBalance)

{

balance = accBalance;

}

**Лістинг interestaccount.h:**

#ifndef INTERESTACCOUNT\_H

#define INTERESTACCOUNT\_H

#include "BankAccount.h"

#include <iostream>

using namespace std;

class InterestAccount : public BankAccount

{

public:

InterestAccount();

InterestAccount(const QString &accNumber,const QString &holder, double bal, double rate);

void setInterestRate(double rate);

double getInterestRate();

void addInterest();

// Перевизначення функцій

void withdraw(double amount) override;

void deposit(double amount) override;

void displayAccountDetails() const override;

private:

double interestRate;

};

#endif // INTERESTACCOUNT\_H

**Лістинг interestaccount.cpp:**

#include "interestaccount.h"

InterestAccount::InterestAccount() : BankAccount(), interestRate(0.0) {}

InterestAccount::InterestAccount(const QString &accNumber, const QString &holder, double bal, double rate)

: BankAccount(accNumber, holder, bal), interestRate(rate)

{

if (rate < 0 || rate > 1)

{

throw invalid\_argument("Interest rate must be between 0 and 1.");

}

}

void InterestAccount::setInterestRate(double rate)

{

if (rate < 0 || rate > 1)

{

throw invalid\_argument("Interest rate must be between 0 and 1.");

}

interestRate = rate;

}

double InterestAccount::getInterestRate()

{

return interestRate;

}

void InterestAccount::addInterest()

{

balance += balance \* interestRate;

}

void InterestAccount::withdraw(double amount)

{

if (amount > balance)

{

throw invalid\_argument("Insufficient balance.");

}

balance -= amount;

}

void InterestAccount::deposit(double amount)

{

if (amount <= 0)

{

throw invalid\_argument("Deposit amount must be positive.");

}

balance += amount;

}

void InterestAccount::displayAccountDetails() const

{

cout << "Interest Account: " << accountNumber.toStdString() << "\n"

<< "Holder: " << accountHolder.toStdString() << "\n"

<< "Balance: " << balance << "\n"

<< "Interest Rate: " << interestRate << "\n";

}

**Лістинг regularaccount.h:**

#ifndef REGULARACCOUNT\_H

#define REGULARACCOUNT\_H

#include "BankAccount.h"

class RegularAccount : public BankAccount

{

public:

RegularAccount() : BankAccount(), minimumBalance(0.0) {}

RegularAccount(const QString &accNumber, const QString &holder, double bal, double minBal);

void setMinBalance(double minBal);

double getMinBalance();

// Перевизначення функцій

void deposit(double amount) override;

void withdraw(double amount) override;

void displayAccountDetails() const override;

private:

double minimumBalance;

};

#endif // REGULARACCOUNT\_H

**Лістинг regularaccount.cpp:**

#include "regularaccount.h"

#include <iostream>

using namespace std;

RegularAccount::RegularAccount(const QString &accNumber, const QString &holder, double bal, double minBal)

: BankAccount(accNumber, holder, bal), minimumBalance(minBal)

{

if (bal < minimumBalance)

{

throw invalid\_argument("Initial balance cannot be less than minimum balance.");

}

}

void RegularAccount::setMinBalance(double minBal)

{

minimumBalance = minBal;

}

double RegularAccount::getMinBalance()

{

return minimumBalance;

}

void RegularAccount::withdraw(double amount)

{

if (balance - amount < minimumBalance)

{

throw invalid\_argument("Withdrawal would breach the minimum balance.");

}

balance -= amount;

}

void RegularAccount::deposit(double amount)

{

if (amount <= 0)

{

throw invalid\_argument("Deposit amount must be positive.");

}

balance += amount;

}

void RegularAccount::displayAccountDetails() const

{

cout << "Regular Account: " << accountNumber.toStdString() << "\n"

<< "Holder: " << accountHolder.toStdString() << "\n"

<< "Balance: " << balance << "\n"

<< "Minimum Balance: " << minimumBalance << "\n";

}

**Висновок:**

В результаті виконання завдань було розроблено дві програми, які демонструють використання об'єктно-орієнтованого програмування в C++ за допомогою фреймворку Qt.

Завдання 1: Гра «Піймай муху»

• Реалізація гри включала створення похідного класу від QMainWindow, що дозволило налаштувати графічний інтерфейс.

• Муха в грі тікала від курсора, що вимагає використання обробки подій, зокрема перевизначення функції event.

• Гра закінчується, коли муха потрапляє в пастку, що продемонструвало реалізацію логіки гри та обробку ігрових подій.

Завдання 2: Система управління банком

• Було розроблено систему, що обробляє різні типи банківських рахунків, реалізуючи наслідування та поліморфізм через базовий абстрактний клас BankAccount.

• Створені похідні класи RegularAccount та InterestAccount містять специфічні характеристики, такі як мінімальний баланс та процентна ставка, реалізуючи відповідні методи для управління балансом.

• Реалізація сетерів і гетерів дозволила контролювати доступ до приватних даних, а обробка виняткових ситуацій забезпечила стабільність програми при некоректних даних (наприклад, при введенні негативної процентної ставки або недостатньому балансі).