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

Звіт
Про виконання лабораторних робіт №9

На тему:

«Принцип поліморфізму»

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: Принцип поліморфізму.
Мета: Навчитись створювати списки об’єктів базового типу, що включають об’єкти похідних типів. Освоїти способи вирішення проблеми неоднозначності при множинному наслідуванні. Вивчити плюси заміщення функцій при множинному наслідуванні. Навчитись використовувати чисті віртуальні функції, знати коли варто використовувати абстрактні класи.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Поліморфізм - одна з трьох основних парадигм ООП. Якщо говорити коротко, поліморфізм – це здатність об'єкта використовувати методи похідного класу, який не існує на момент створення базового.
Уявімо ситуацію, що ми створюємо програму, в якій ми працюємо з класами тварин. Один з цих класів Bird(птахи), а другий Mammal(ссавці). Клас Bird містить функцію Fly(), а клас Mammal містить функцію Gallop() – біг галопом.
А тепер сталось так, що нам потрібно створити новий міфічний персонаж – крилатого Пегаса (Pegasus), який буде гібридом між птахом та ссавцем. В таких випадках використовують множинне наслідування.
Тепер ми зможемо пройтись по списку птахів та в кожного елемента списку викликати метод Fly() а в списку ссавців відповідно метод Gallop(). Тут ми і використовуємо принцип поліморфізму.
Для того щоб викликати функцію з класу Pegasus нам потрібно привести вказівник базового типу до похідного типу. Для цього використовуємо функцію dynamic_cast. Якщо вказівник не вдалось привести до похідного типу функція верне нуль.
Заміщення функції вирішує дві проблеми:
Зникає невизначеність звертання до базових класів;
Функцію можна замістити таким чином, що в похідному класі при виклику цієї функції викликати функцію з базового класу.
Клас, який містить чисті віртуальні функції є абстрактним. Неможливо створити об’єкт абстрактного класу. Поміщення в клас чистої віртуальної функції означає наступне:
- неможливо створити об’єкт цього класу;
- необхідно замістити чисту віртуальну функцію в похідному класі.
Будь-який клас, наслідування від абстрактного класу, наслідує від нього чисту віртуальну функцію, яку необхідно замістити щоб отримати можливість створювати об’єкти цього класу.

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

1. Розробити ієрархію класів відповідно до варіанту.
2. Використати множинне наслідування, продемонструвати вирішення проблеми з неоднозначністю доступу до членів базових класів за допомогою віртуального наслідування, за допомогою явного звертання до членів класу та за допомогою заміщення функцій в похідному класі (при потребі).

3. Створити списки об’єктів базового типу, в них помістити об’єкти похідного типу. Продемонструвати виклик функцій з об’єктів – елементів списку. Використати оператор dynamic_cast (при потребі).
4. Створити абстрактний клас, використати чисто віртуальну функцію, що містить реалізацію в базовому класі.
5. Для вивільнення динамічної пам’яті використовувати віртуальні деструктори.
6. Сформувати звіт до лабораторної роботи. Відобразити в ньому діаграму наслідування класів.

Завдання згідно варіанту

Варіант 2

Лабіринт будівельник

Тут те саме. Базовий клас-інтерфейс MazeBuilder. Його реалізації: SimpleMazeBuilder,
MiddleMazeBuilder, ComplexMazeBuilder. Можна зробити, щоб сам об’єкт Maze міг виводити
себе якось на форму.

Код програми

mazebuilder.h

```
#ifndef MAZEBUILDER_H
#define MAZEBUILDER_H

#include <QString>
#include <QLabel>

class Maze {
public:
 virtual void createMaze(QLabel* lbl) = 0;
 virtual ~Maze(){}
};

class MazeBuilder : public Maze {
public:
 MazeBuilder(){}

 virtual ~MazeBuilder(){}


 void setShape(QString str){
   this->shape = str;
 }
 void setStyle(QString str){
   this->style = str;
 }
 void setSize(int s){
   this->size = s;
```

```
 }

 QString getShape(){ return this->shape; }
 virtual void createMaze(QLabel* lbl) override{
  lbl->setPixmap(QPixmap("://mazes//Maze//RectangularOrthogonal20.png"));
 }

protected:
 QString shape;
 QString style;
 int size;
};

class MiddleMazeBuilder : public virtual MazeBuilder {
public:
 virtual ~MiddleMazeBuilder(){}

 MiddleMazeBuilder(){}
 MiddleMazeBuilder(QString shape, int size){
  this->shape = shape;
  this->size = size;
 }

 virtual void createMaze(QLabel* lbl) override{
  lbl->setPixmap(QPixmap("://mazes//Maze//"+shape+QString::number(size)+".png"));
 }
};

class ComplexMazeBuilder : public virtual MazeBuilder {
public:
 virtual ~ComplexMazeBuilder(){}

 ComplexMazeBuilder(QString shape){this->shape = shape;}
 ComplexMazeBuilder(QString shape, QString style, int size){
  this->shape = shape;
  this->style = style;
  this->size = size;
 }

 virtual void createMaze(QLabel* lbl) override{
  QString str;
  str = ((getShape()) == "Rectangular") ? style : "";
  lbl->setPixmap(QPixmap("://mazes//Maze//"+shape+str+QString::number(size)+".png"));
 }
};

class SimpleMazeBuilder : public ComplexMazeBuilder, public MiddleMazeBuilder {
public:
 virtual ~SimpleMazeBuilder(){}

 SimpleMazeBuilder(QString str):ComplexMazeBuilder(str){}
 virtual void createMaze(QLabel* lbl) override{
  QString str;
  str = ((getShape()) == "Rectangular") ? "RectangularOrthogonal" : getShape();
  lbl->setPixmap(QPixmap("://mazes//Maze//"+str+"20.png"));
```

```
 }
};
#endif // MAZEBUILDER_H
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "mazebuilder.h"

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_comboBox_type_activated(int index);

 void on_comboBox_form_activated(int index);

 void on_pushButton_clicked();

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

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

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

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

```
void MainWindow::on_comboBox_type_activated(int index)
{
 switch(index){
  case(0):{
     ui->comboBox_size->setEnabled(false);
     ui->comboBox_rect_type->setEnabled(false);
     break;
    }
  case(1):{
     ui->comboBox_size->setEnabled(true);
     ui->comboBox_rect_type->setEnabled(false);
     break;
    }
  case(2):{
     ui->comboBox_rect_type->setEnabled(true);
     break;
    }
 }
}


void MainWindow::on_comboBox_form_activated(int index)
{
  if(index != 0){
     ui->comboBox_rect_type->setEnabled(false);
   } else {
     ui->comboBox_rect_type->setEnabled(true);
    }
}


void MainWindow::on_pushButton_clicked()
{
  int currClass = ui->comboBox_type->currentIndex();
  int currSize = ui->comboBox_size->currentIndex();
  int size;
  switch(currSize){
    case(0): size = 10; break;
    case(1): size = 20; break;
    case(2): size = 30; break;
    default: size = 20;
    }

  SimpleMazeBuilder* m0 = new
SimpleMazeBuilder(ui->comboBox_form->currentText());
  MiddleMazeBuilder* m1 = new MiddleMazeBuilder(ui->comboBox_form->currentText(),
size);
  ComplexMazeBuilder* m2 = new
ComplexMazeBuilder(ui->comboBox_form->currentText(),
ui->comboBox_rect_type->currentText() , size);
  MazeBuilder* mazelist[3];
  mazelist[0] = m0;
  mazelist[1] = m1;
  mazelist[2] = m2;
  if(currClass == 0){
```

```
        mazelist[0]->createMaze(ui->label);
    } else if(currClass == 1){

        mazelist[1]->createMaze(ui->label);
    } else if(currClass == 2){

        mazelist[2]->createMaze(ui->label);
    }

  for(int i = 0; i < 3; i++){
      delete mazelist[i];
    }
}
```

Вигляд програми

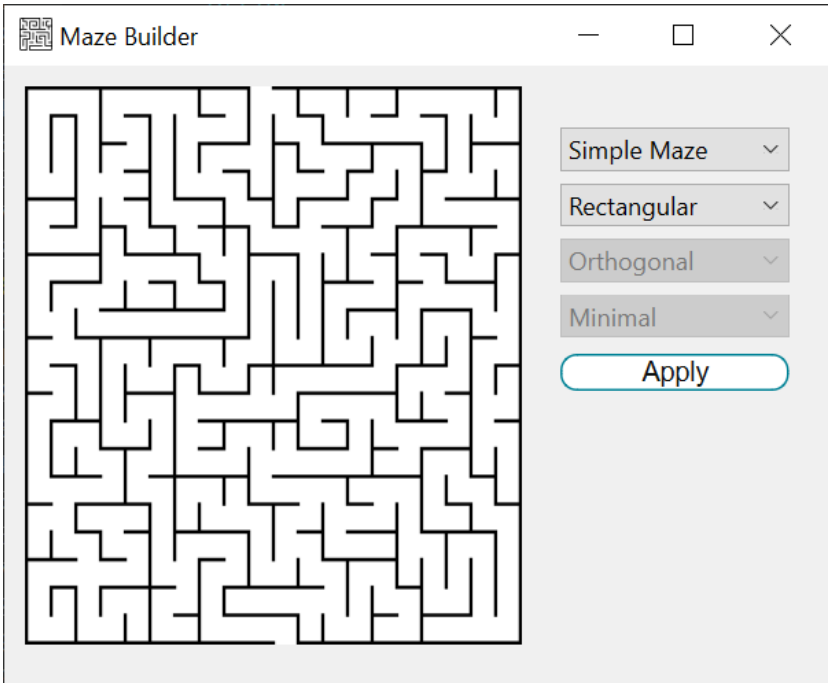

Рис 1. Приклад об’єкту SimpleMazeBuilder

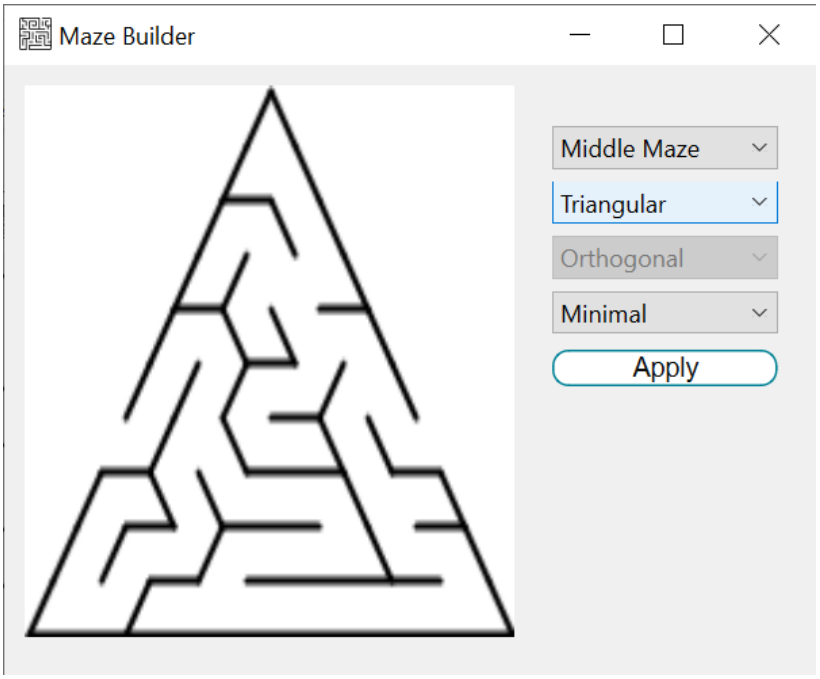

Рис 2. Приклад об’єкту MiddleMazeBuilder

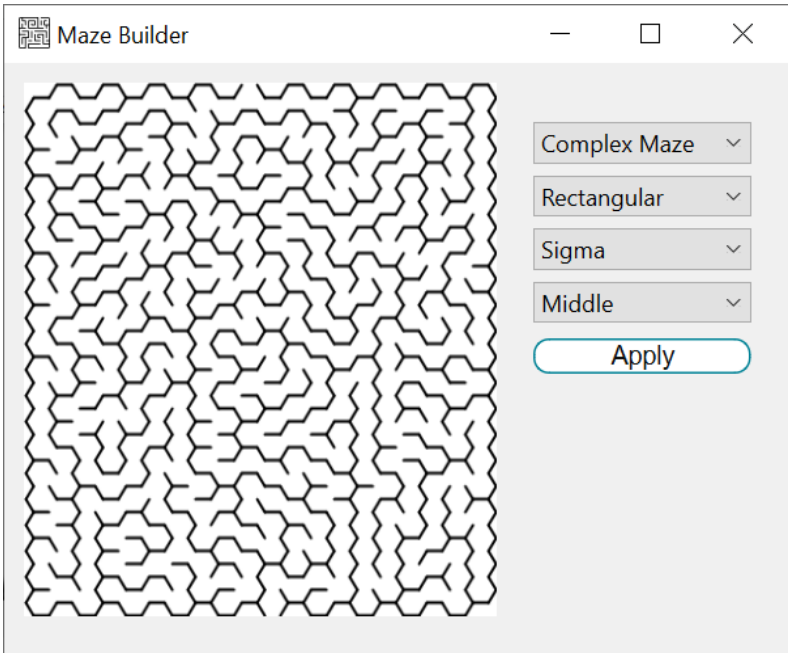

Рис 2. Приклад об’єкту ComplexMazeBuilder

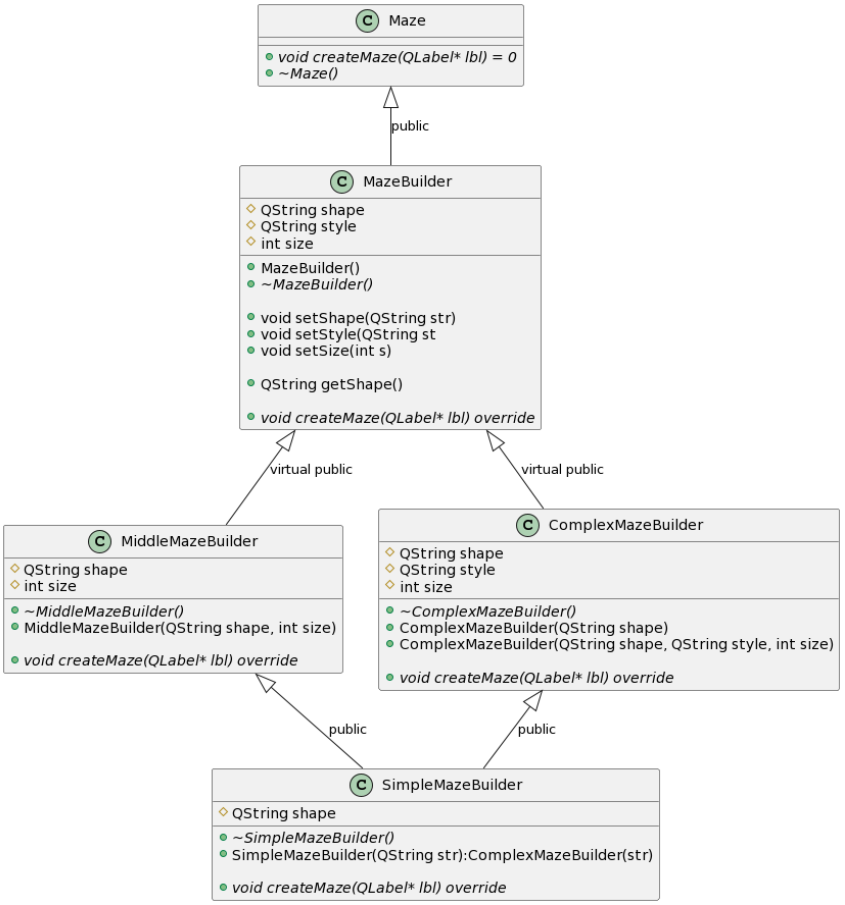

Рис 3.UML-діаграма

ВИСНОВКИ

Виконавши лабораторну роботу №9 я навчився створювати списки об’єктів базового типу, що включають об’єкти похідних типів. Освоїв способи вирішення проблеми неоднозначності при множинному наслідуванні. Вивчив плюси заміщення функцій при множинному наслідуванні. Навчився використовувати чисті віртуальні функції, дізнався коли варто використовувати абстрактні класи.