МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

ЗВІТ

До лабораторної роботи № 9
На тему: «Принцип поліморфізму»
З дисципліни: «Об'єктно-орієнтоване програмування»

Лектор:
доц. кафедри ПЗ
Коротєєва Т. О.

Виконав:
ст. гр. ПЗ-16
Чаус О. М.

Прийняв:
асист. кафедри ПЗ
Дивак І. В.

« ____ » _____ 2022 р.

∑= _____ .....:

Львів-2022

Тема роботи: Принцип поліморфізму

Мета роботи: 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.

Для того щоб викликати функцію з похідного класу 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.

Проблеми множинного наслідування:

Можлива наступна ситуація – клас наслідує батьківський декілька разів. Способи вирішення:

1. Явно звертатись до членів потрібного нам батьківського класу.
2. Використання віртуального наслідування.

Абстрактні типи даних

В ооп часто створюються ієрархії логічно пов’язаних класів. Наприклад уявимо клас Shape, від якого наслідуються класи Rectangle і Circle. Пізніше від класу Rectangle наслідується клас Square, як окремий вид прямокутників. В класі Shape немає змісту визначати функції знаходження площі чи знаходження периметру, оскільки в похідних класах вони будуть перевизначені. Всі методи повинні функціонувати нормально в похідних класах а не в базовому класі Shape, оскільки неможливо створити екземпляр форми як такий. Також програма повинна бути захищена від спроби користувача створити об’єкт цього класу.

C++ підтримує створення абстрактних типів даних з чистими віртуальними функціями. Чисті віртуальні функції це такі, які ініціалізуються нульовим значенням, наприклад

```
virtual void Draw() = 0;
```

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, наслідуваний від абстрактного класу, наслідує від нього чисту віртуальну функцію, яку необхідно замістити щоб отримати можливість створювати об’єкти цього класу.

Виконання чистих віртуальних функцій

Зазвичай чисті віртуальні функції оголошуються в абстрактному базовому класі і не виконуються. Оскільки неможливо створити об’єкт абстрактного базового класу, як правило, нема необхідності і в виконання чистої віртуальної функції.

Тим не менше, інколи виникає необхідність виконання чистої віртуальної функції. Вона може бути викликана з об’єкта, який наслідує абстрактний клас, наприклад, щоб забезпечити загальну функціональність для всіх заміщених функцій.

Завдання для 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 опратор 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.

Хід роботи

Файл **array.h**:

```
#ifndef ARRAY_H
#define ARRAY_H


class Array
{
protected:
    double *m_array;
    unsigned m_size;
public:
    Array(unsigned size);
    virtual ~Array();
    virtual double& operator[](unsigned index) = 0;
    virtual unsigned getSize() const = 0;
    virtual double getMax() const = 0;
    virtual double getMin() const = 0;
    virtual Array& add(double val) = 0;
    virtual Array& subtractIndex(unsigned index) = 0;
};


class SimpleArray : virtual public Array
{
public:
    SimpleArray(unsigned size) : Array(size) {};
    double& operator[](unsigned index) override;
    unsigned getSize() const override;
    double getMax() const override;
    double getMin() const override;
    Array& add(double val) override;
    Array& subtractIndex(unsigned index) override;
};


class StatisticsArrayInterface : virtual public Array
{
public:
    StatisticsArrayInterface(unsigned size) : Array(size) {};
    virtual double min() = 0;
    virtual double max() = 0;
    virtual double average() = 0;
    virtual double median() = 0;
};

class StatisticsArray : public SimpleArray, public StatisticsArrayInterface
{
```

```
public:
    StatisticsArray(unsigned size) : Array(size), SimpleArray(size),
StatisticsArrayInterface(size) {};
    double min();
    double max();
    double average();
    double median();
};

#endif // ARRAY_H
```

Файл **array.cpp**:

```
#include "array.h"

Array::Array(unsigned size) : m_size(size)
{
    m_array = new double[size];
    for(int i = 0; i < m_size; i++)
    {
        m_array[i] = 0;
    }
}

Array::~Array()
{
    delete[] m_array;
}

unsigned SimpleArray::getSize() const
{
    return this->m_size;
}
double& SimpleArray::operator[](unsigned index)
{
    return m_array[index];
}
double SimpleArray::getMax() const
{
    double max_value = m_array[0];
    for(int i = 0; i < this->m_size; i++)
        if(m_array[i] > max_value) max_value = m_array[i];
    return max_value;
}
double SimpleArray::getMin() const
{
    double min_value = m_array[0];
    for(int i = 0; i < this->m_size; i++)
        if(m_array[i] < min_value) min_value = m_array[i];
    return min_value;
}
Array& SimpleArray::add(double val)
{
    m_size++;
    SimpleArray tempArray(m_size);
    for(int i = 0 ; i < m_size - 1; i++)
        tempArray[i] = m_array[i];
    tempArray[m_size - 1] = val;
    delete[] m_array;
    m_array = new double[m_size];
    for(int i = 0 ; i < m_size ; i++)
        m_array[i] = tempArray[i];
    return * this;
}
Array& SimpleArray::subtractIndex(unsigned index)
{
    if(m_size)
    {
```

```
        m_size--;
        SimpleArray tempArray(m_size + 1);
        for(int i = 0 ; i < m_size + 1; i++)
            tempArray[i] = m_array[i];
        delete[] m_array;
        m_array = new double[m_size];
        for(int i = 0 ; i < m_size ; i++)
            if(i < index)
                m_array[i] = tempArray[i];
            else m_array[i] = tempArray[i + 1];
    }
    return *this;
}

double StatisticsArray::min()
{
    int minCount = 0;
    double element = 0;
    int elementCount = 0;
    for(int i = 0; i < m_size; i++)
    {
        for(int j = 0; j < m_size; j++)
        {
            if(m_array[j] == m_array[i])
                elementCount++;
        }
        if(minCount == 0 || minCount > elementCount)
        {
            element = m_array[i];
            minCount = elementCount;
        }
        elementCount = 0;
    }
    return element;
}
double StatisticsArray::max()
{
    int maxCount = 0;
    double element = 0;
    int elementCount = 0;
    for(int i = 0; i < m_size; i++)
    {
        for(int j = 0; j < m_size; j++)
        {
            if(m_array[j] == m_array[i])
                elementCount++;
        }
        if(maxCount < elementCount)
        {
            element = m_array[i];
            maxCount = elementCount;
        }
        elementCount = 0;
    }
    return element;
}
double StatisticsArray::average()
{
    double sum = 0;
    for(int i = 0; i < m_size; i++)
        sum += m_array[i];
    return sum / m_size;
}
double StatisticsArray::median()
{
    if(!(m_size % 2))
```

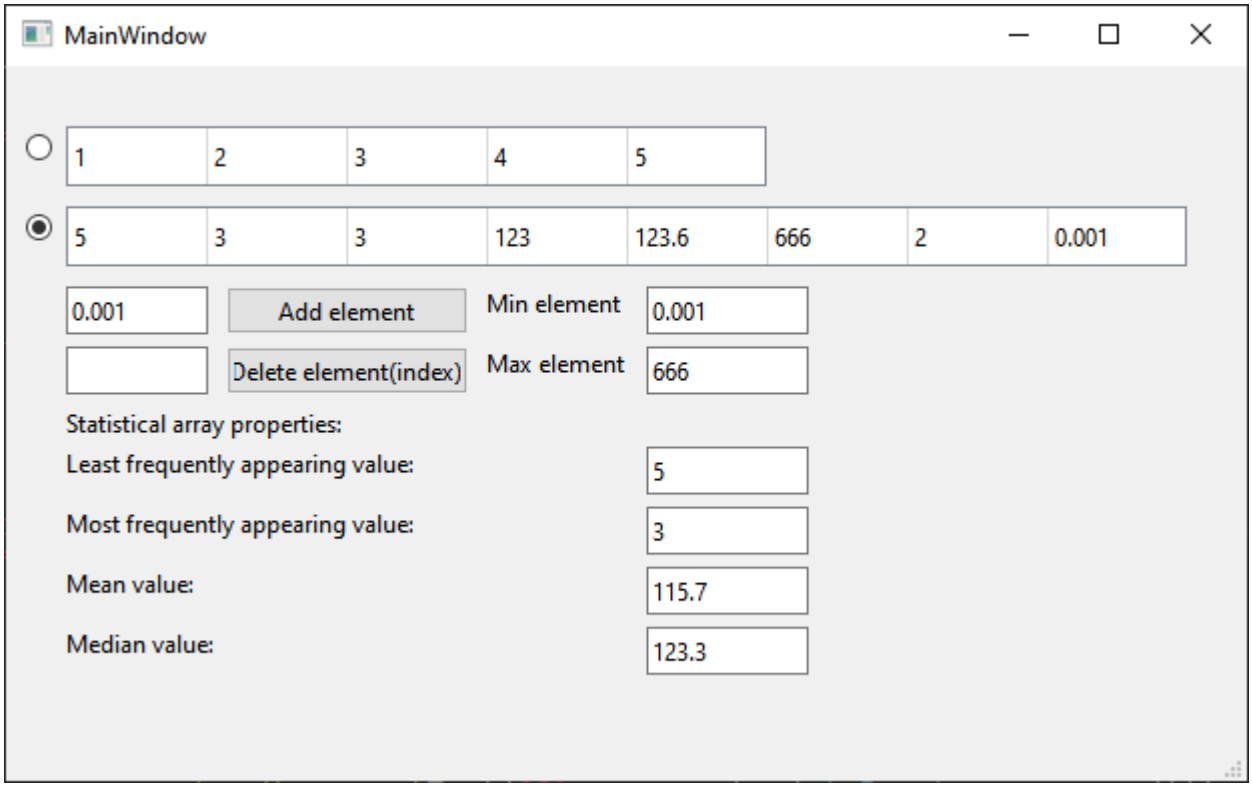
```
    {
        return (m_array[m_size / 2] + m_array[(m_size / 2) - 1]) / 2;
    }
    return m_array[(m_size / 2)];
}
```

Файл **mainwindow.cpp**:

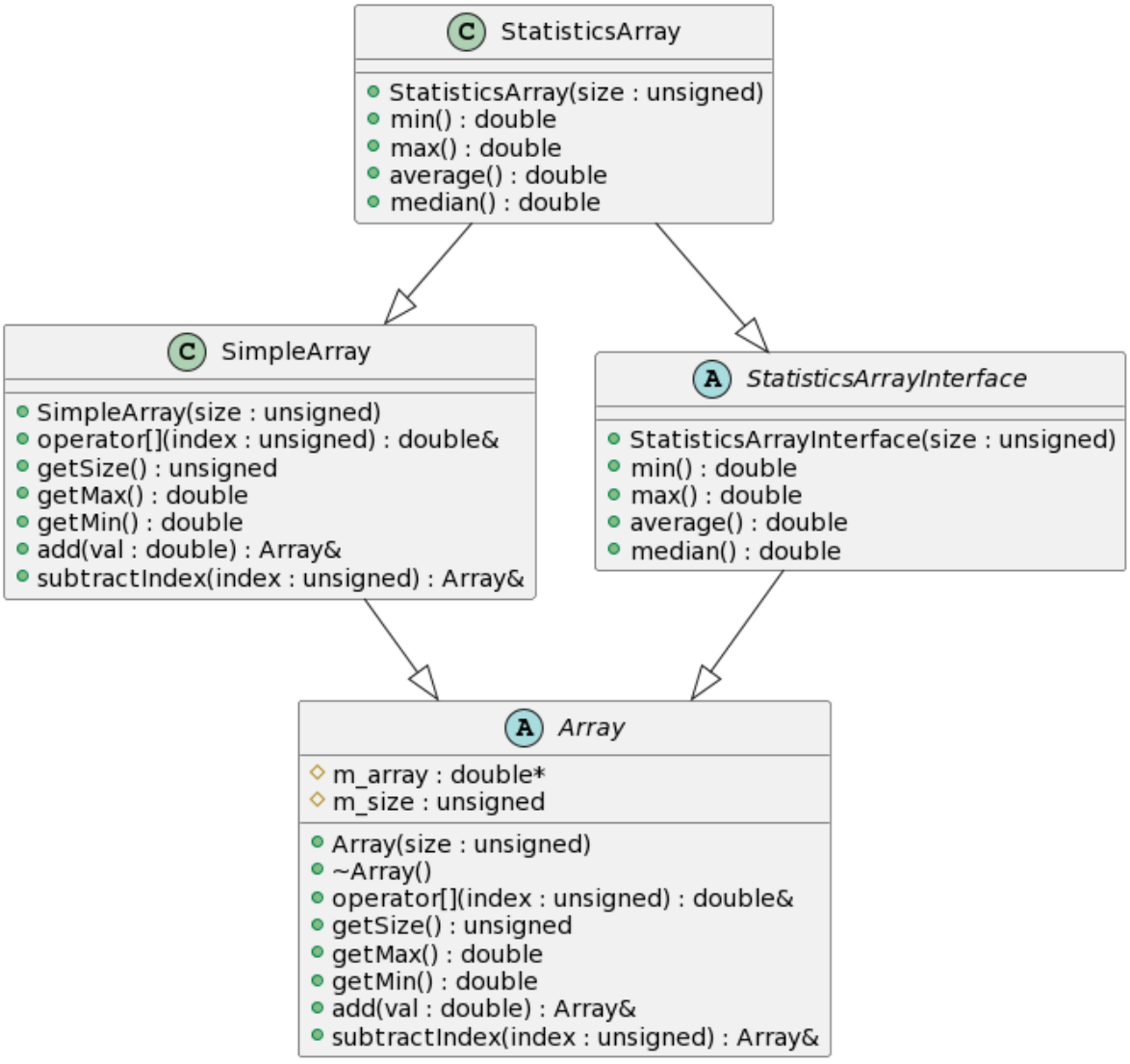
```
Array *array1 = new SimpleArray(0);
Array *array2 = new StatisticsArray(0);
Array *list[] = {array1, array2};

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

MainWindow::~MainWindow()
{
    for (Array *x : list)
           delete x;
    delete ui;
}

void MainWindow::updateData()
{
    ui->Array1Table->setColumnCount(list[0]->getSize());
    ui->Array1Table->resize(ui->Array1Table->columnCount() * 70, ui->Array1Table-
>height());
    for(int i = 0; i < list[0]->getSize(); i++)
    {
        QTableWidgetItem *item = new QTableWidgetItem;
        item->setFlags(item->flags() ^ Qt::ItemIsEditable);
        item->setText(QString::number((*list[0])[i]));
        ui->Array1Table->setItem(0, i, item);
    }
    ui->Array2Table->setColumnCount(list[1]->getSize());
    ui->Array2Table->resize(ui->Array2Table->columnCount() * 70, ui->Array2Table-
>height());
    for(int i = 0; i < list[1]->getSize(); i++)
    {
        QTableWidgetItem *item = new QTableWidgetItem;
        item->setFlags(item->flags() ^ Qt::ItemIsEditable);
        item->setText(QString::number((*list[1])[i]));
        ui->Array2Table->setItem(0, i, item);
    }
    if(ui->array1button->isChecked())
    {
        if(list[0]->getSize())
        {
            ui->lineMax->setText(QString::number(list[0]->getMax()));
            ui->lineMin->setText(QString::number(list[0]->getMin()));
        }
        else
        {
            ui->lineMax->setText("No elements");
            ui->lineMin->setText("No elements");
        }
         StatisticsArray *staPointer = dynamic_cast<StatisticsArray *>(list[0]);
        if(staPointer)
        {
            if(staPointer->getSize())
            {
```

```
            ui->lineLeast->setText(QString::number(staPointer->min()));
            ui->lineMost->setText(QString::number(staPointer->max()));
            ui->lineMean->setText(QString::number(staPointer->average()));
            ui->lineMedian->setText(QString::number(staPointer->median()));
        }
        else
        {
            ui->lineLeast->setText("No elements");
            ui->lineMost->setText("No elements");
            ui->lineMean->setText("No elements");
            ui->lineMedian->setText("No elements");
        }
    }
    else
    {
        ui->lineLeast->setText("");
        ui->lineMost->setText("");
        ui->lineMean->setText("");
        ui->lineMedian->setText("");
    }
}
}
if(ui->array2button->isChecked())
{
    if(list[0]->getSize())
    {
        ui->lineMax->setText(QString::number(list[0]->getMax()));
        ui->lineMin->setText(QString::number(list[0]->getMin()));
    }
    else
    {
        ui->lineMax->setText("No elements");
        ui->lineMin->setText("No elements");
    }
    StatisticsArray *staPointer = dynamic_cast<StatisticsArray *>(list[1]);
    if(staPointer)
    {
        if(staPointer->getSize())
        {
            ui->lineLeast->setText(QString::number(staPointer->min()));
            ui->lineMost->setText(QString::number(staPointer->max()));
            ui->lineMean->setText(QString::number(staPointer->average()));
            ui->lineMedian->setText(QString::number(staPointer->median()));
        }
        else
        {
            ui->lineLeast->setText("No elements");
            ui->lineMost->setText("No elements");
            ui->lineMean->setText("No elements");
            ui->lineMedian->setText("No elements");
        }
    }
    else
    {
        ui->lineLeast->setText("");
        ui->lineMost->setText("");
        ui->lineMean->setText("");
        ui->lineMedian->setText("");
    }
}
else
{
    ui->lineLeast->setText("");
    ui->lineMost->setText("");
    ui->lineMean->setText("");
    ui->lineMedian->setText("");
```

```
    }
}

void MainWindow::on_bAddElement_clicked()
{
    if(!ui->lineValue->text().isEmpty())
    {
        if(ui->array1button->isChecked())
        {
            list[0]->add(ui->lineValue->text().toDouble());
        }
        else if(ui->array2button->isChecked())
        {
            list[1]->add(ui->lineValue->text().toDouble());
        }
        updateData();
    }
}


void MainWindow::on_bAddElement_2_clicked()
{
    if(!ui->lineIndex->text().isEmpty())
    {
        if(ui->array1button->isChecked())
        {
            if(!(ui->lineIndex->text().toInt() > list[0]->getSize() - 1) && !(ui-
>lineIndex->text().toInt() < 0))
            {
                list[0]->subtractIndex(ui->lineIndex->text().toInt());
            }
        }
        else if(ui->array2button->isChecked())
        {
            if(!(ui->lineIndex->text().toInt() > list[1]->getSize() - 1) && !(ui-
>lineIndex->text().toInt() < 0))
            {
                list[1]->subtractIndex(ui->lineIndex->text().toInt());
            }
        }
        updateData();
    }
}


void MainWindow::on_array1button_clicked()
{
    updateData();
}


void MainWindow::on_array2button_clicked()
{
    updateData();
}
```



Зображення програми



Діаграма класів

Висновок: : 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, знати коли варто використовувати абстрактні класи.