**ПРАКТИЧЕСКАЯ РАБОТА № 1.1**

**Тема:** Использование принципов ООП: инкапсуляции, наследования, полиморфизма.

**Цель:** Научиться использовать основные принципы ООП при разработке приложений.

**Время выполнения:** 6 часов.

**I. Входной контроль.**

1) Инкапсуляция – принцип ООП, суть которого является разграничение данных в классе.

Наследование – принцип ООП, в котором классы могут быть базовыми и производными, где производные классы наследуют методы и поля базового класса.

Полиморфизм – принцип ОПП, который ставит программисту задачу обобщения. Этот прицнип можно описать так: «Один интерфейс, множество методов».

2) Конструктор – метод, вызываемый при создании экземпляра класса.

Деструктор – метод, вызываемый при явном или неявном разрушении экземпляра класса.

Модификатор доступа private – член класса, помеченный данным модификатором может быть доступен только изнутри класса.

Модификатор доступа pulibc – член класса, помеченный данным модификатором может быть доступен отовсюду.

Модификатор доступа protected – член класса, помеченный данным модификатором может быть доступен производным классом от данного.

**II. Ход выполнения работы**

В ходе выполнения работы был создан абстрактный класс интерфейс, в котором были описаны все нужные свойства и методы, производный от этого класса класс, реализовывал функци абстрактного класса. Так же был создан класс MyApplication, который реализовывал меню для управления классом Matrix. Так же каждое изменение было сопровождено commit’ами и итог работы опубликован на GitHub.

**Листинг 1 – исходный код модуля MatrixInterface.h**

{========================================================}

{Модуль MatrixInterface }

{группа П-32 }

{Разработчик: Ромашко М.В. }

{Модифицирован: 13 марта 2018 }

{--------------------------------------------------------}

{Создан для обобщения }

{========================================================}

#include <iostream>

#include <ctime>

using namespace std;

// Interface for inheriance

#pragma once

class MatrixInterface

{

protected:

int \*\*array; // Prototype for dynamic array

int rowCount; // Count of rows

int colCount; // Count of columns

public:

MatrixInterface(); // Default constructor

MatrixInterface(int colCount, int rowCount); // Constructor with params

MatrixInterface(const MatrixInterface& m); // Copy constructor

~MatrixInterface(); // Destructor

virtual void Print() = 0; // Purely virtual function for matrix print

virtual void ChangeCol(int count) = 0; // Purely virtual function for changing column count

virtual void ChangeRow(int count) = 0; // Purely virtual function for changing row count

virtual void PrintSubMatr(int c, int r) = 0; // Purely virtual function for printing fixed matrix

virtual void SetRandom() = 0; // Purely virtual function for random filling a matrix

};

**Листинг 2 – исходный код модуля MatrixInterface.cpp**

{========================================================}

{Модуль MatrixInterface }

{группа П-32 }

{Разработчик: Ромашко М.В. }

{Модифицирован: 13 марта 2018 }

{--------------------------------------------------------}

{Создан для реализации конструкторов }

{========================================================}

// Why not??

#include "stdafx.h"

#include "MatrixInterface.h"

MatrixInterface::MatrixInterface()

{

}

MatrixInterface::MatrixInterface(int colCount, int rowCount)

{

}

MatrixInterface::MatrixInterface(const MatrixInterface & m)

{

}

MatrixInterface::~MatrixInterface()

{

}

**Листинг 3 – исходный код модуля Matrix.h**

{========================================================}

{Модуль Matrix }

{группа П-32 }

{Разработчик: Ромашко М.В. }

{Модифицирован: 13 марта 2018 }

{--------------------------------------------------------}

{Шаблон для конкретной реализации Matrix }

{========================================================}

/\*

Realization of class MatrixInterface

\*/

#include "MatrixInterface.h"

#pragma once

class Matrix : public MatrixInterface

{

public:

Matrix(); // Default constructor

Matrix(int colCount, int rowCount); // Constructor with params

Matrix(const Matrix &m); // Copy constructor

~Matrix(); // Destructor

/\*

In these functions we tell the compiler that we are redefining purely virtual functions

defined in interface

\*/

virtual void SetRandom() override;

virtual void Print() override;

virtual void ChangeCol(int count) override;

virtual void ChangeRow(int count) override;

virtual void PrintSubMatr(int c, int r) override;

};

**Листинг 4 – исходный код модуля Matrix.cpp**

{========================================================}

{Модуль Matrix }

{группа П-32 }

{Разработчик: Ромашко М.В. }

{Модифицирован: 13 марта 2018 }

{--------------------------------------------------------}

{Конкретная реализации Matrix }

{========================================================}

#include "stdafx.h"

#include "Matrix.h"

Matrix::Matrix() // Так как размера нету, просто включаем рандом

{

srand((unsigned)time(0));

}

Matrix::Matrix(int colCount, int rowCount) // Размер есть, создаём массив размера соответствующий параметрам

{

srand((unsigned)time(0));

this->colCount = colCount;

this->rowCount = rowCount;

this->array = new int\*[rowCount];

for (int i = 0; i < rowCount; i++)

{

this->array[i] = new int[colCount];

}

this->SetRandom(); // Рандомим его

}

Matrix::Matrix(const Matrix & m) // Копируем массив и свойства из массива, который в параметрах

{

this->colCount = m.colCount;

this->rowCount = m.rowCount;

this->array = m.array;

}

void Matrix::SetRandom() // Рандомим

{

for (int i = 0; i < this->rowCount; i++)

{

for (int j = 0; j < this->colCount; j++)

{

this->array[i][j] = rand() % 30 - 10;

}

}

}

void Matrix::Print() // Выводим массив

{

for (int i = 0; i < this->rowCount; i++)

{

for (int j = 0; j < this->colCount; j++)

{

cout << this->array[i][j] << " ";

}

cout << endl;

}

}

void Matrix::ChangeCol(int count) // Изменяем количество колонок

{

this->colCount = count;

for (int i = 0; i < rowCount; i++)

{

this->array[i] = new int[colCount];

}

this->SetRandom();

}

void Matrix::ChangeRow(int count) // Изменяем количество строк

{

this->rowCount = count;

this->array = new int\*[rowCount];

for (int i = 0; i < rowCount; i++)

{

this->array[i] = new int[colCount];

}

this->SetRandom();

}

void Matrix::PrintSubMatr(int c, int r) // Выводим подматрицу размером из параметров

{

for (int i = 0; i < r; i++)

{

for (int j = 0; j < c; j++)

{

cout << this->array[i][j] << " ";

}

cout << endl;

}

}

Matrix::~Matrix() // Динамически освобождаем память от массива

{

delete[] this->array;

}

**Листинг 5 – исходный код модуля MyApplication.h**

{========================================================}

{Модуль MyApplication }

{группа П-32 }

{Разработчик: Ромашко М.В. }

{Модифицирован: 13 марта 2018 }

{--------------------------------------------------------}

{Шаблон для конкретной реализации меню для управления }

{Классом Matrix }

{========================================================}

/\*

Class-menu for controlling class Matrix

\*/

#include "Matrix.h"

#pragma once

class MyApplication

{

public:

MyApplication();

~MyApplication();

};

**Листинг 6 – исходный код модуля MyApplication.cpp**

{========================================================}

{Модуль MyApplication }

{группа П-32 }

{Разработчик: Ромашко М.В. }

{Модифицирован: 13 марта 2018 }

{--------------------------------------------------------}

{Конкретная реализация меню для управления классом Matrix}

{========================================================}

#include "stdafx.h"

#include "MyApplication.h"

MyApplication::MyApplication()

{

Matrix \*m = new Matrix(0, 0); // initializing empty matrix

char input = 0; // variable, wich stores the user input

int keyCode = 0, // variable input with int type

c = 0, // For changing column count

r = 0; // For changing row count

while ((int)input != 48) // While zero not inputed

{

cout << "1 - Create a array" << endl

<< "2 - Print a array" << endl

<< "3 - Change column count" << endl

<< "4 - Change row count" << endl

<< "5 - Print the sub matrix" << endl

<< "0 - Exit" << endl;

cin >> input;

keyCode = (int)input; // Char to int

switch (keyCode)

{

case 49: // Matrix init

{

cout << "Input column and row count: "; cin >> c >> r;

m = new Matrix(c, r);

break;

}

case 50: // Printing matrix

{

m->Print();

break;

}

case 51: // Changing count of the cols

{

cout << "Input new column count: "; cin >> c;

m->ChangeCol(c);

break;

}

case 52: // Changing count of the rows

{

cout << "Input new row count: "; cin >> r;

m->ChangeRow(r);

break;

}

case 53: // Printing the submatr

{

cout << "Input the size of sub matrix: "; cin >> c >> r;

m->PrintSubMatr(c, r);

break;

}

}

system("pause");

system("cls");

}

}

MyApplication::~MyApplication()

{

}

**III. Выводы**

В результате выполнения работы были изучены 3 принципа ООП и применены на практике, так как в данном проекте присутствует наследование, разграничение методов, абстракция как частный случай полиморфизма. Так же укреплены навыки пользования git.