**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

};