Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: Виртуальные функции. Полиморфизм.

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Токарев Павел Аркадьевич

Проверил

Доцент кафедры ИТАС

Полякова Ольга Андреевна

Г. Пермь-2023

**Постановка задачи**

1.Определить абстрактный класс.

2.Сформировать иерархию классов (лабораторная работа №4).

3.Определить класс вектор, элементами которого будут объекты иерархии классов.

4.Перегрузить операции ввода и вывода

**Анализ**

1.Определим класс Pair, представляющий собой пару чисел.

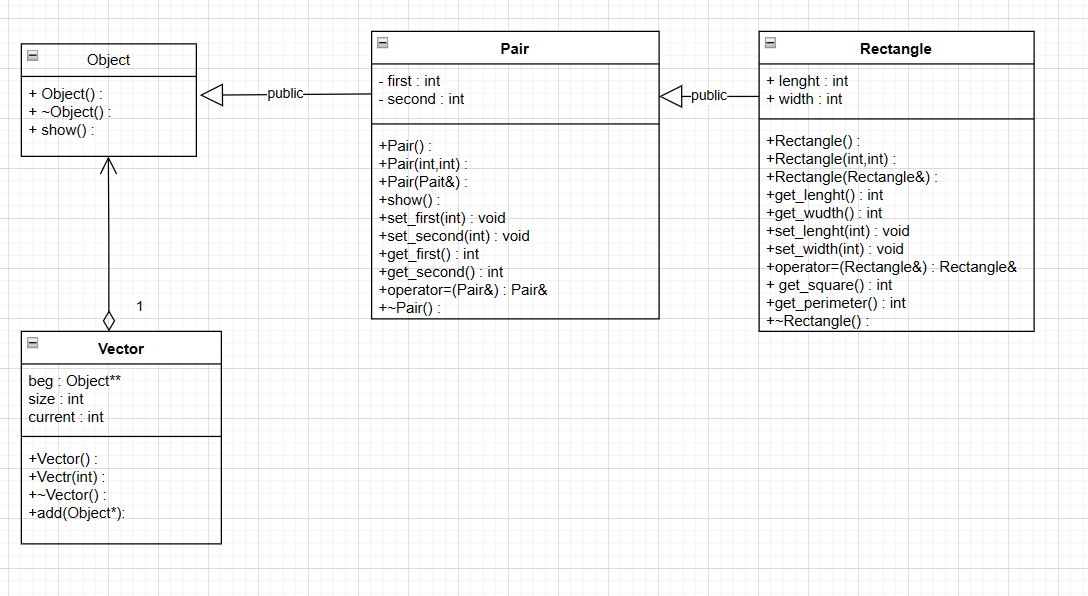
2.От класса Pair определим производный класс Rectangle.

3.Перегрузим операции ввода и вывода.

4.Определим класс Vector, который содержит поля size,current и \*\*beg.

5. size отвечает за размер вектора, current за текущий элемент, а \*\*beg указывает на динамический массив, хранящий в себе указатель на объекты классов из иерархии классов.

**Диаграмма UML**

****

**Код программы**

main.cpp

#include<iostream>

#include"Object.h"

#include"Pair.h"

#include"Rectangle.h"

#include"Vector.h"

using namespace std;

int main() {

setlocale(LC\_ALL, "rus");

Object\* p;

cout << "Прямоугольник a"<<endl;

Rectangle a;

cin >> a;

cout << "Пара чисел b" << endl;

Pair b;

cin >> b;

cout << "Выведем вектор c" << endl;

Vector c(2);

p = &a;

c.add(p);

p = &b;

c.add(p);

cout << c<<endl;

}

Pair.h

#pragma once

#include<iostream>

#include"Object.h"

using namespace std;

class Pair:

public Object

{

public:

Pair(void);

Pair(int, int);

Pair(const Pair&);

void show();

virtual ~Pair(void);

int get\_first();

int get\_second();

void set\_first(int);

void set\_second(int);

Pair& operator=(const Pair&);

friend istream& operator>>(istream& in, Pair& t);

friend ostream& operator<<(ostream& out, const Pair& t);

int multyply();

private:

int first;

int second;

};

Pair.cpp

#include "Pair.h"

#include<iostream>

using namespace std;

void Pair::show() {

cout << "Первое число: " << first << endl;

cout << "Второе число: " << second << endl;

}

Pair::Pair(void) {

first = 0;

second = 0;

}

Pair::Pair(int tmp1, int tmp2) {

first = tmp1;

second = tmp2;

}

Pair::Pair(const Pair& t) {

first = t.first;

second = t.second;

}

Pair::~Pair(void) {

};

int Pair::get\_first() {

return first;

}

int Pair::get\_second() {

return second;

}

void Pair::set\_first(int tmp) {

first = tmp;

}

void Pair::set\_second(int tmp) {

second = tmp;

}

Pair& Pair::operator=(const Pair& t) {

first = t.first;

second = t.second;

return \*this;

}

istream& operator>>(istream& in, Pair& t) {

cout << "Первое число:";

in >> t.first;

cout << "Второе число:";

in >> t.second;

return in;

}

ostream& operator<<(ostream& out, const Pair& t) {

out << "Первое число:" << t.first;

out << "Второе число:" << t.second;

out << endl;

return out;

}

int Pair::multyply() {

return(first \* second);

}

Rectangle.h

#pragma once

#include"Pair.h"

using namespace std;

class Rectangle :

public Pair

{

public:

Rectangle(void);

public:

~Rectangle(void);

Rectangle(int,int,int,int);

Rectangle(const Rectangle& t);

int get\_leght();

int get\_width();

void set\_lenght(int);

void set\_width(int);

Rectangle& operator=(const Rectangle&);

friend istream& operator>>(istream& in, Rectangle& t);

friend ostream& operator<<(ostream& out, const Rectangle& t);

int get\_square();

int get\_perimeter();

void show();

private:

int lenght;

int width;

};

Rectangle.cpp

#include "Rectangle.h"

#include<iostream>

using namespace std;

Rectangle::Rectangle(void) :Pair() {

lenght = 0;

width = 0;

}

Rectangle::Rectangle(int tmp1,int tmp2,int tmp3,int tmp4) :Pair(tmp1,tmp2) {

lenght = tmp3;

width = tmp4;

}

Rectangle::Rectangle(const Rectangle& t) {

width = t.width;

lenght = t.lenght;

}

void Rectangle::set\_lenght(int t) {

lenght = t;

}

int Rectangle::get\_leght() {

return lenght;

}

Rectangle& Rectangle::operator=(const Rectangle& t) {

lenght = t.lenght;

width = t.width;

return \*this;

}

int Rectangle::get\_perimeter() {

return(2 \* lenght + 2 \* width);

}

int Rectangle::get\_square() {

return(lenght \* width);

}

Rectangle::~Rectangle(void) {

}

void Rectangle::show() {

cout << "Длина : " << lenght<<endl;

cout << "Ширина : " << width << endl;

}

istream& operator>>(istream& in, Rectangle& t) {

cout << "Длина : ";

in >> t.lenght;

cout << "Ширина : ";

in >> t.width;

return in;

}

ostream& operator<<(ostream& out,const Rectangle& t) {

out << "Длина : ";

out << t.lenght<<endl;

out << "Ширина : ";

out << t.width<<endl;

return out;

}

Object.h

#pragma once

#include<iostream>

using namespace std;

class Object

{

public:

Object(void);

public:

~Object(void);

virtual void show() = 0;

};

Object.cpp

#include "Object.h"

void Object::show() {

}

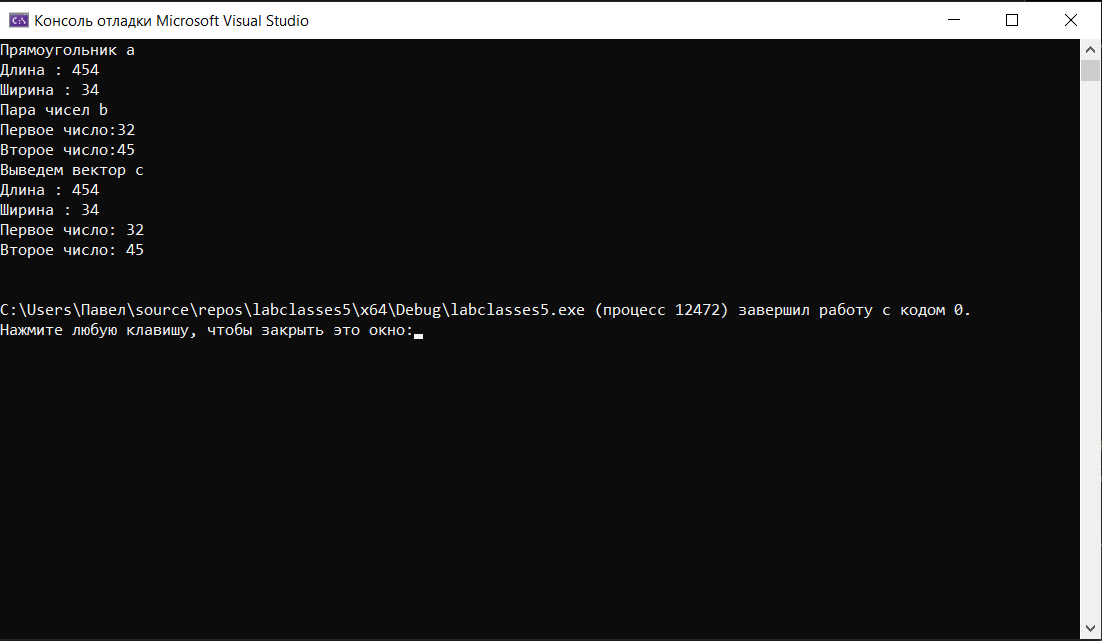
Object::Object() {

}

Object::~Object() {

}

**Результат работы программы**

****

**Анализ результатов**

Удалось успешно реализовать иерархию наследования классов. Перегруженный операторы класса контейнера исправно возвращают хранимые в них экземпляры классов.

**Контрольные вопросы**

1. Виртуальная функция-это функция-член базового класса, которая может быть переопределена производным классом. Чистая виртуальная функция-это функция-член базового класса, единственное объявление которой содержится в базовом классе и должно быть определено в производном классе, иначе производный класс также становится абстрактным (содержит =0 вместо тела).

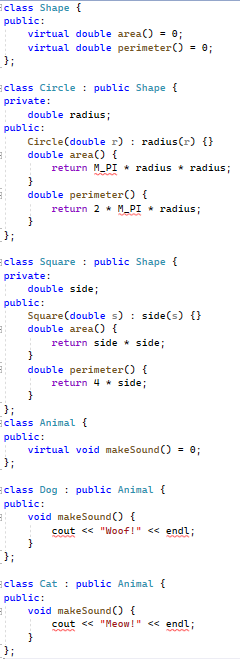
2.Абстрактный класс-класс, содержащий хотя бы один чисто виртуальный метод.

3.Абстрактные классы предназначены для предоставления общих представлений, которые конкретизируются в производных классах.

4.Полиморфные функции-функции, работающие с объектами любого типа в рамках одной иерархии классов.

5. Полиморфизм - это способность объектов разных классов использовать одинаковый интерфейс. Принцип подстановки - это возможность использовать объект производного класса вместо объекта базового класса.

6.В обоих примерах абстрактный класс («Shape» и «Animal») определяет интерфейс, который должны реализовать производные классы («Circle», «Square», «Dog», «Cat»). Это позволяет создавать общие методы и свойства, которые будут доступны для всех производных классов.



7.Примеры полиморфных функций:



8.Механизм позднего связывания используется при работе с объектами разных классов, которые могут иметь одинаковый интерфейс. Он позволяет вызывать соответствующий метод в зависимости от типа объекта во время выполнения программы, а не во время компиляции. Это особенно полезно при работе с абстрактными классами и их производными классами.