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: Лабораторная работа по классам №5

Выполнил работу  
Студент группы РИС-22-1Б  
Ишемцева Мария Андреевна  
Проверил  
Доцент кафедры ИТАС  
Полякова Ольга Андреевна

Пермь 2023

Постановка задачи:

1. Определить абстрактный класс.
2. Определить иерархию классов, в основе которой будет находиться абстрактный класс (см. лабораторную работу №4).
3. Определить класс Вектор, элементами которого будут указатели на объекты иерархии классов.
4. Перегрузить для класса Вектор операцию вывода объектов с помощью потоков.
5. В основной функции продемонстрировать перегруженные операции и полиморфизм Вектора.

Анализ задачи:

Определен абстрактный класс object, имеющий чистую виртуальную ф-ию show(). Пользовательский класс-наследник – Person. Пользовательский класс-наследник – Student. Объекты обоих классов-наследников записаны в отдельный пользовательский класс Vector и выведены на экран с помощью потокового вывода.

Ответы на вопросы:

1. Чисто виртуальному присваивается значение 0. Класс становится абстрактным.
2. Абстрактный класс – класс, содержащий в себе хотя бы 1 чистую виртуальную ф-ию.
3. Для дальнейшей реализации указанных методов в классах-наследниках.
4. Способность функции обрабатывать разные типы данных.
5. При полиморфизме ф-ия может изменять свой функционал.

class Figure

{

public:

    virtual double getSquare() =0;

    virtual double getPerimeter() =0;

    virtual void showFigureType()=0;

};

class Rectangle : public Figure

{

private:

    double width;

    double height;

public:

    Rectangle(double w, double h) : width(w), height(h)

    {

    }

    double getSquare() override

    {

        return width \* height;

    }

    double getPerimeter() override

    {

        return width \* 2 + height \* 2;

    }

    void showFigureType()

    {

        std::cout << "Rectangle" << std::endl;

    }

};

class Circle : public Figure

{

private:

    double radius;

public:

    Circle(double r) : radius(r)

    {

    }

    double getSquare() override

    {

        return radius \* radius \* 3.14;

    }

    double getPerimeter() override

    {

        return 2 \* 3.14 \* radius;

    }

    void showFigureType()

    {

        std::cout << "Circle" << std::endl;

    }

};

int main()

{

    Rectangle rect(30, 50);

    Circle circle(30);

    std::cout << "Rectangle square: " << rect.getSquare() << std::endl;

    std::cout << "Circle square: " << circle.getSquare() << std::endl;

    return 0;

}

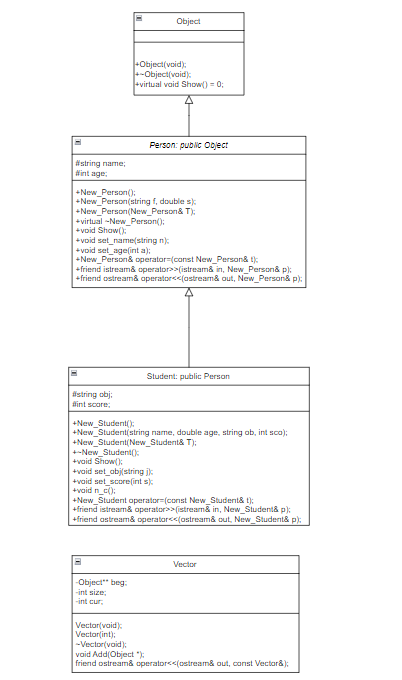
1. class Abstract{ //Абстрактный класс

public:virtual void print\_msg()=0;

};

1. Позднее связывание означает, что объект связывается с вызовом функции только во время ис­полнения программы, а не раньше. Позднее связывание достигается в С++ с помощью использо­вания виртуальных функций и производных классов.

UML Диаграмма:



Код программы:

#include "Vector.h"

#include <string>

#include <iostream>

using namespace std;

void main()

{

setlocale(0, "RUS");

New\_Person a;

cin >> a;

cout << a << endl;//вывод с помощью перегруженной операции

Object\* p = &a;

p->Show();

New\_Student b;

cin >> b;

cout << b << endl;//вывод с помощью перегруженной операции

p = &b;

p->Show(); //вывод с помощью метода Show() и указателя

Vector v(5);//вектор из 5 элементов

cin >> a;

cin >> b;

p = &a;//ставим указатель на объект класса Car

v.Add(p);//добавляем объект в вектор

p = &b;//ставим указатель на объект класса Lorry

v.Add(p); //добавляем объект в вектор

cout << v;//вывод вектора

}

#include "Vector.h"

//конструктор без параметров

Vector::Vector(void) {

beg = 0;

size = 0;

cur = 0;

}

//деструктор

Vector::~Vector(void) {

if (beg != 0)delete[] beg;

beg = 0;

}

//конструктор с параметрами

Vector::Vector(int n) {

beg = new Object \* [n];

cur = 0;

size = n;

}

//добавление объекта, на который указывает указатель р в вектор

void Vector::Add(Object \* p)

{

if (cur < size)

beg[cur] = p;

cur++;

}

//операция вывода

ostream& operator<<(ostream& out, const Vector& v) {

if (v.size == 0) out << "Empty" << endl;

Object\*\* p = v.beg;//указатель на указатель типа Object

for (int i = 0; i < v.cur; i++)

{

(\*p)->Show();//вызов метода Show() (позднее связывание)

p++;//передвигаем указатель на следующий объект

return out;

}

}

#pragma once

#include "New\_Student.h"

#include <string>

#include <iostream>

using namespace std;

class Vector {

Object\*\* beg;

int size;

int cur;

public:

Vector(void);

Vector(int);

~Vector(void);

void Add(Object \*);

friend ostream& operator<<(ostream& out, const Vector&);

};

#include "Object.h"

Object::Object(void) {

}

Object::~Object(void) {

}

#pragma once

class Object {

public:

Object(void);

~Object(void);

virtual void Show() = 0;

};

#include <iostream>

#include "New\_Student.h"

//#include "Person.h"

using namespace std;

New\_Student::New\_Student() {

obj = "";

score = 0;

cout << "Конструктор без параметров для " << this << endl;

}

New\_Student::New\_Student(string name, double age, string ob, int sco) :New\_Person(name, age) {

obj = ob;

score = sco;

cout << "Коснтруктор с параметрами для - " << this << endl;

}

New\_Student::New\_Student(New\_Student& T) {

name = T.name;

age = T.age;

obj = T.obj;

score = T.score;

cout << "Конструктор копирования для - " << this << endl;

}

New\_Student::~New\_Student() {

cout << "Деструктор для - " << this << endl;

}

void New\_Student::Show() {

cout << "Имя: " << name;

cout << "\nВозраст: " << age;

cout << "\nПредмет: " << obj;

cout << "\nОценка: " << score << endl;

}

void New\_Student::set\_obj(string j) {

obj = j;

}

void New\_Student::set\_score(int s) {

score = s;

}

void New\_Student::n\_c() {

if (score >= 3) {

cout << "Оценка удовлетворительная или выше\n";

}

else {

cout << "Оценка неудовлетворительная\n";

}

}

New\_Student New\_Student::operator=(const New\_Student& t)

{

//проверка на самоприсваивание

if (&t == this) return \*this;

name = t.name;

age = t.age;

obj = t.obj;

score = t.score;

return \*this;

}

istream& operator>>(istream& in, New\_Student& p) {

cout << "name? "; in >> p.name;

cout << "age? "; in >> p.age;

cout << "obj? "; in >> p.obj;

cout << "score? "; in >> p.score;

return in;

}

ostream& operator<<(ostream& out, New\_Student& p)

{

out << "Имя: " << p.name << endl;

out << "Возраст: " << p.age << endl;

out << "Предмет: " << p.obj << endl;

out << "Оценка: " << p.score << endl;

return out;

}

#include <iostream>

#include "New\_Person.h"

using namespace std;

class New\_Student :

public New\_Person {

public:

New\_Student();

New\_Student(string name, double age, string ob, int sco);

New\_Student(New\_Student& T);

~New\_Student();

void Show();

void set\_obj(string j);

void set\_score(int s);

void n\_c();

New\_Student operator=(const New\_Student& t);

friend istream& operator>>(istream& in, New\_Student& p);

friend ostream& operator<<(ostream& out, New\_Student& p);

protected:

string obj;

int score;

};

#include <iostream>

#include "Object.h"

using namespace std;

class New\_Person: public Object

{

public:

New\_Person();

New\_Person(string f, double s);

New\_Person(New\_Person& T);

virtual ~New\_Person();

void Show();

void set\_name(string n);

void set\_age(int a);

New\_Person& operator=(const New\_Person& t);

friend istream& operator>>(istream& in, New\_Person& p);

friend ostream& operator<<(ostream& out, New\_Person& p);

protected:

string name;

int age;

};

#include <iostream>

#include "New\_Person.h"

using namespace std;

New\_Person::New\_Person() {

name = "";

age = 0;

cout << "Конструктор без параметров для " << this << endl;

}

New\_Person::New\_Person(string f, double s) {

name = f;

age = s;

cout << "Коснтруктор с параметрами для - " << this << endl;

}

New\_Person::New\_Person(New\_Person& T) {

name = T.name;

age = T.age;

cout << "Конструктор копирования для - " << this << endl;

}

New\_Person::~New\_Person() {

cout << "Деструктор для - " << this << endl;

}

void New\_Person::Show() {

cout << "Имя: " << name;

cout << "\nВозраст: " << age << endl;

}

void New\_Person::set\_name(string n) {

name = n;

}

void New\_Person::set\_age(int a) {

age = a;

}

New\_Person& New\_Person::operator=(const New\_Person& t)

{

//проверка на самоприсваивание

if (&t == this) return \*this;

name = t.name;

age = t.age;

return \*this;

}

istream& operator>>(istream& in, New\_Person& p) {

cout << "name? "; in >> p.name;

cout << "age? "; in >> p.age;

return in;

}

ostream& operator<<(ostream& out, New\_Person& p)

{

return (out << "Имя: " << p.name << endl << "Возраст: " << p.age << endl);

}

Скриншоты результатов:

