Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**о работе по информатике**

**Тема:** Лабораторная работа № 5 по ООП

Семестр: 2

Выполнил студент ИВТ-22-2б:

Солодов Александр Андреевич

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.
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.

*Вариант 12*

Базовый класс: ЧЕЛОВЕК (PERSON)

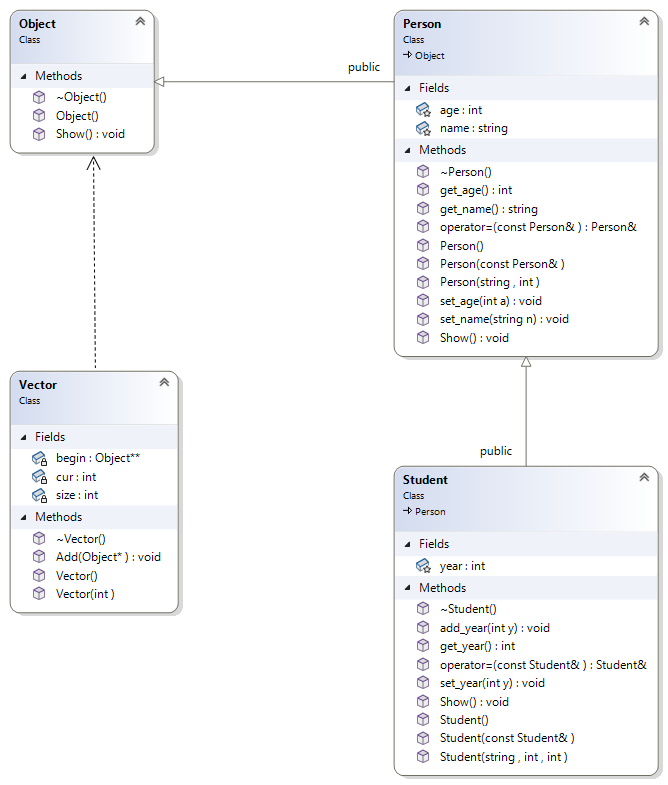
Имя (name) – string

Возраст (age) – int

Определить методы изменения полей.

Создать производный класс STUDENT, имеющий поле год обучения. Определить методы изменения и увеличения года обучения.

**UML-диаграмма**

****

**Программное решение**

Object.h

#pragma once

class Object {

public:

Object();

~Object();

virtual void Show() = 0; //чисто виртуальная функция

};

Object.cpp

#include "Object.h"

Object::Object() {}

Object::~Object() {}

Person.h

#pragma once

#include <iostream>

#include <string>

#include "Object.h"

using namespace std;

class Person : public Object

{

protected:

string name;

int age;

public:

Person();

Person(string, int);

Person(const Person&);

~Person();

string get\_name() { return name; }

void set\_name(string n) { name = n; }

int get\_age() { return age; }

void set\_age(int a) { age = a; }

void Show();

Person& operator = (const Person&);

friend istream& operator >> (istream& in, Person& p);

friend ostream& operator << (ostream& out, const Person& p);

};

Person.cpp

#include "Person.h"

Person::Person()

{

name = "";

age = 0;

}

Person::Person(string n, int a)

{

name = n;

age = a;

}

Person::Person(const Person& p)

{

name = p.name;

age = p.age;

}

Person::~Person()

{

//cout << "Вызван деструктор для класса Person: " << this << endl;

}

void Person::Show() {

cout << "Имя: " << name << endl;

cout << "Возраст: " << age << endl;

}

Person& Person::operator = (const Person& p) {

if (&p == this)

return \*this;

name = p.name;

age = p.age;

return \*this;

}

istream& operator >> (istream& in, Person& p)

{

cout << "\nВведите имя: ";

in >> p.name;

cout << "Введите возраст: ";

in >> p.age;

return in;

}

ostream& operator << (ostream& out, const Person& p)

{

out << p.name << " | " << p.age << endl;

return out;

}

Student.h

#pragma once

#include "Person.h"

class Student : public Person {

protected:

int year;

public:

Student();

Student(string, int, int);

Student(const Student&);

~Student();

//методы класса

int get\_year() { return year; }

void set\_year(int y) { year = y; }

void add\_year(int y) { year += y; }

void Show();

Student& operator = (const Student&);

//перегрузка операций ввода и вывода

friend istream& operator >> (istream& in, Student& s);

friend ostream& operator << (ostream& out, const Student& s);

};

Student.cpp

#include "Student.h"

Student::Student() : Person()

{

year = 0;

}

Student::Student(string n, int a, int y) : Person(n, a)

{

year = y;

}

Student::Student(const Student& s)

{

name = s.name;

age = s.age;

year = s.year;

}

Student::~Student()

{

//cout << "Вызван деструктор для класса Student: " << this << endl;

}

void Student::Show() {

cout << "Имя: " << name << endl;

cout << "Возраст: " << age << endl;

cout << "Год поступления: " << year << endl;

}

Student& Student::operator = (const Student& s) {

if (&s == this)

return \*this;

name = s.name;

age = s.age;

year = s.year;

return \*this;

}

istream& operator >> (istream& in, Student& s)

{

cout << "\nВведите имя: ";

in >> s.name;

cout << "Введите возраст: ";

in >> s.age;

cout << "Введите год поступления: ";

in >> s.year;

return in;

}

ostream& operator << (ostream& out, const Student& s)

{

out << s.name << " | " << s.age << " | " << s.year << endl;

return out;

}

Vector.h

#pragma once

#include "Object.h"

#include <iostream>

#include <string>

using namespace std;

class Vector {

private:

Object\*\* begin; //указатель на первый элемент вектора

int size; //размер

int cur; //текущая позиция

public:

Vector(); //конструктор без параметров

Vector(int); //конструктор c параметрами

~Vector(); //деструктор

void Add(Object\*); //добавление элемента в вектор

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

};

Vector.cpp

#include "Vector.h"

//конструктор без параметров

Vector::Vector(void) {

begin = 0;

size = 0;

cur = 0;

}

//деструктор

Vector::~Vector(void) {

if (begin != 0) {

delete[] begin;

}

begin = 0;

}

//конструктор с параметрами

Vector::Vector(int n) {

begin = new Object \* [n];

cur = 0;

size = n;

}

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

void Vector::Add(Object\* p) {

if (cur < size) {

begin[cur] = p;

cur++;

}

}

//операция вывода

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

if (v.size == 0) out << "Вектор пуст!" << endl;

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

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

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

cout << "----------" << endl;

p++; //передвигаем указатель на следующий объект

}

return out;

}

main.cpp

#include "Vector.h"

#include "Object.h"

#include "Student.h"

#include "Person.h"

#include <string>

#include <iostream>

using namespace std;

int main() {

setlocale(LC\_ALL, "RU");

Vector v(5); //создание вектора

Person p1; //создание и заполнение объекта класса Person

cin >> p1;

Student s1("Melissa", 22, 2008); //создание и заполнение объекта класса Student

Object\* ptr = &p1; //установка указателя на объект класса Person

v.Add(ptr); //добавление объекта в вектор

ptr = &s1; //установка указателя на объект класса Student

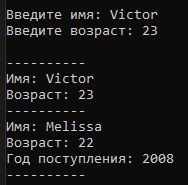
v.Add(ptr); //добавление объекта в вектор

cout << endl << "----------" << endl << v; //вывод вектора на экран

return 0;

}

**Результат работы программы**

****

**Контрольные вопросы**

*1. Какой метод называется чисто виртуальным? Чем он отличается от виртуального метода?*

Чисто виртуальный метод – это функция без определения или реализации. Виртуальный метод (виртуальная функция) — это метод класса, который может быть переопределён в классах-наследниках так, что конкретная реализация метода для вызова будет определяться во время исполнения.

*Пример чисто виртуального метода:*

virtual void Show() = 0;

*2. Какой класс называется абстрактным?*

Абстрактный класс – базовый класс, который не предполагает создания экземпляров. Класс, содержащий хотя бы один Чисто виртуальный метод, будет [абстрактным](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BB%D0%B0%D1%81%D1%81).

*Пример:*

class Object

{

public:

Object();

~Object();

virtual void Show() = 0; //Чисто виртуальная функция

};

*3. Для чего предназначены абстрактные классы?*

Абстрактные классы предназначены для создания обобщенных  сущностей, на основе которых в дальнейшем предполагается создавать более конкретные производные классы.

*Пример:*

Нужно создать для класса TextFile базовый класс File, от которого будет унаследован еще один класс RTFFile. Однако, в такой ситуации неизвестно как реализовать метод read() класса File, т.к. класс File не реализует поведение какого-то конкретного типа файлов, а представляет интерфейс для работы с различными файлами. Для этого используется чисто виртуальная функция.

class File {

virtual string read(int count) = 0;

};

Метод read(...) должен быть определен в классах наследниках. Теперь класс File стал абстрактным, и его экземпляры невозможно создать. Но можно работать через указатель на абстрактный класс с объектами производных классов.

File\* f = new TextFile("text.txt");

//различные действия с файлом text.txt

delete f;

f = new RTFFile("rich\_text.rtf");

//различные действия с файлом rich\_text.rtf

delete f;

*4. Что такое полиморфные функции?*

При работе с абстрактными классами можно создать функцию, параметром которой будет являться указатель на абстрактный класс. Туда может передаваться указатель на объект любого производного класса, что позволяет создать полиморфные функции, то есть функции, работающие с объектом любого типа в пределах одной иерархии.

class Base {

virtual void work() = 0;

};

class Derived : public Base { ... };

void print(Base\* a) { cout << "Hello!"; }

int main() {

Derived\* t = new Derived;

print(t); //Функция работает, хотя в качестве параметра передается

//указатель на объект не абстрактного класса

return 0;

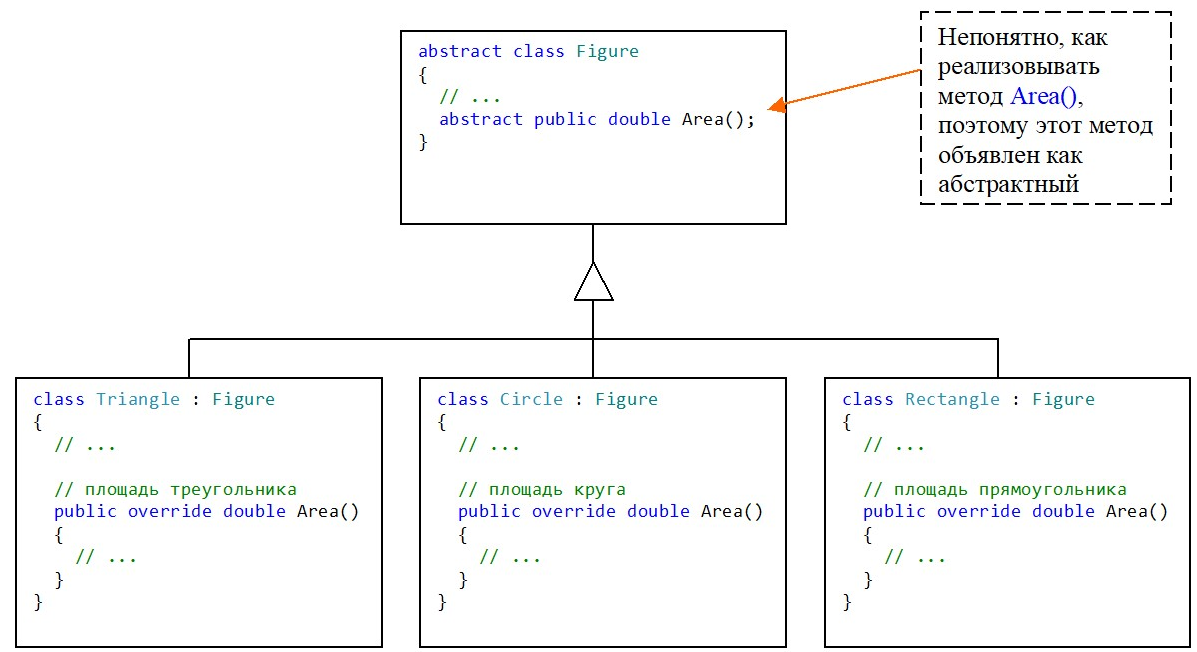
*5. Чем полиморфизм отличается от принципа подстановки?*

Полиморфизм - возможность иметь разные формы для одной и той же сущности.

Принцип подстановки: функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа не зная об этом.

Например, если у нас есть класс A и унаследованный от него класс B, то если мы заменим все использования класса A на B, ничего не должно измениться в работе программы. Ведь класс B всего лишь расширяет функционал класса A.

*6. Привести примеры иерархий с использованием абстрактных классов.*



*7. Привести примеры полиморфных функций.*

Вопрос 4: функция void print(Base\* a) { cout << "Hello!"; }

*8. В каких случаях используется механизм позднего связывания?*

В случаях вызова полиморфной функции с типом параметра класса-наследника.