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**

**ОТЧЕТ**

**о работе по информатике**

Семестр: 2

На тему: «ООП. Наследование. Виртуальные функции. Полиморфизм»

Выполнил студент ИВТ-22-2б:

Казанцев Антон Васильевич

(дата, подпись)

Проверила:

Полякова Ольга Андреевна

(дата, подпись)

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 15:*

Базовый класс: ЧЕЛОВЕК (PERSON)

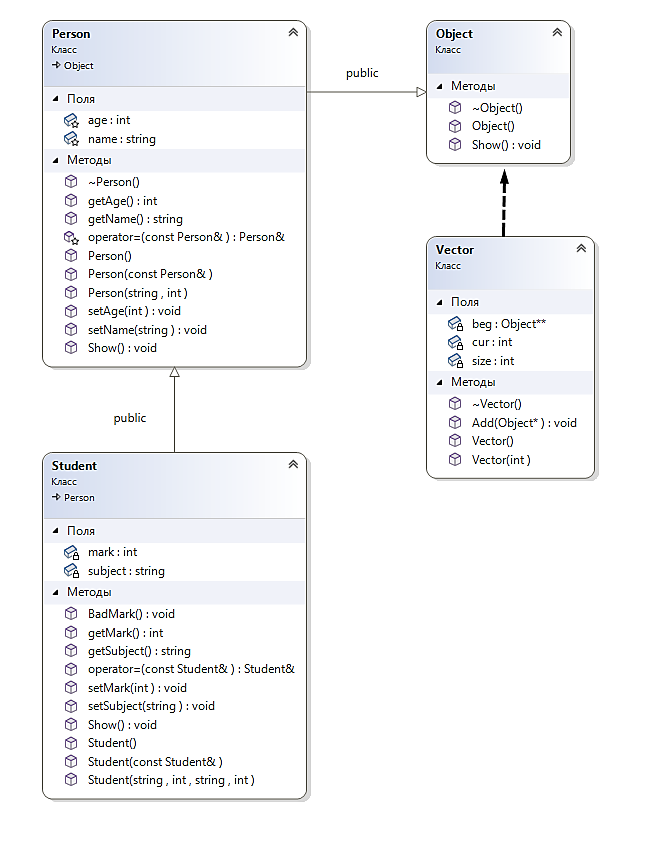
Имя (name) – string

Возраст (age) – int

Определить методы изменения полей.

Создать производный класс STUDENT, имеющий поля Предмет – string и Оценка – int. Определить методы изменения полей и метод, выдающий сообщение о неудовлетворительной оценке.

**UML-Диаграмма**



**Программное решение**

***Object.h***

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Object

{

public:

Object() {};

~Object() {};

virtual void Show() = 0;

};

***Vector.h***

#pragma once

#include "Object.h"

class Vector {

public:

Vector();

Vector(int);

~Vector();

void Add(Object\*);

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

private:

Object\*\* beg;

int size;

int cur;

};

***Vector.cpp***

#include "Vector.h"

Vector::Vector() {

beg = 0;

size = 0;

cur = 0;

}

Vector::~Vector() {

if (beg != 0) {

delete[] beg;

}

beg = 0;

}

Vector::Vector(int n) {

beg = new Object \* [n];

cur = 0;

size = n;

}

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

if (cur < size) {

beg[cur] = p;

cur++;

}

}

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

if (v.size == 0) {

out << "Вектор пуст!\n";

}

Object\*\* p = v.beg;

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

(\*p)->Show();

p++;

}

return out;

}

***Student.h***

#pragma once

#include <iostream>

#include "Person.h"

using namespace std;

class Student : public Person {

private:

string subject;

int mark;

public:

void Show();

Student();

Student(string, int, string, int);

Student(const Student&);

void setSubject(string);

void setMark(int);

string getSubject();

int getMark();

void BadMark();

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

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

Student& operator=(const Student&);

};

***Student.cpp***

#include "Student.h"

void Student::Show() {

cout << name << ' ' << age << ' ' << subject << ' ' << mark << '\n';

}

Student::Student() {

name = "";

subject = "";

age = 0;

mark = 0;

}

Student::Student(string name, int age, string subject, int mark) {

this->name = name;

this->age = age;

this->subject = subject;

this->mark = mark;

}

Student::Student(const Student& copied) {

name = copied.name;

age = copied.age;

subject = copied.subject;

mark = copied.mark;

}

void Student::setSubject(string subject) {

this->subject = subject;

}

void Student::setMark(int mark) {

this->mark = mark;

}

string Student::getSubject() {

return subject;

}

int Student::getMark() {

return mark;

}

void Student::BadMark() {

if (mark < 4) {

cout << this->name << " имеет плохую оценку\n";

}

}

istream& operator>>(istream& in, Student& cur) {

cout << "Введите данные в формате:\nимя возвраст предмет оценка\n";

return (in >> cur.name >> cur.age >> cur.subject >> cur.mark);

}

ostream& operator<<(ostream& out, const Student& cur) {

return (out << cur.name << ' ' << cur.age << ' ' << cur.subject << ' ' << cur.mark << '\n');

}

Student& Student::operator=(const Student& m\_student) {//перегрузка оператора присваивания

if (&m\_student != this) {

name = m\_student.name;

age = m\_student.age;

subject = m\_student.subject;

mark = m\_student.mark;

}

return \*this;

}

***Person.h***

#pragma once

#include <iostream>

#include <string>

#include "Object.h"

using namespace std;

class Person : public Object {

protected:

string name;

int age;

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

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

Person& operator=(const Person&);

public:

void Show();

void setName(string);

void setAge(int);

string getName();

int getAge();

Person();

Person(string, int);

Person(const Person&);

~Person() {};

};

***Person.cpp***

#include "Person.h"

Person::Person() {

this->name = "";

this->age = 0;

}

Person::Person(string name, int age) {

this->name = name;

this->age = age;

}

Person::Person(const Person& copied) {

age = copied.age;

name = copied.name;

}

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

cout << "Введите данные в формате:\nимя возвраст\n";

return (in >> p.name >> p.age);

}

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

return (out << p.name << ' ' << p.age << '\n');

}

void Person::Show() {

cout << name << ' ' << age << '\n';

}

void Person::setName(string name) {

this->name = name;

}

void Person::setAge(int age) {

this->age = age;

}

string Person::getName() {

return name;

}

int Person::getAge() {

return age;

}

Person& Person::operator=(const Person& m\_person) {//перегрузка оператора присваивания

if (&m\_person != this) {

name = m\_person.name;

age = m\_person.age;

}

return \*this;

}

***LabaOOP5.cpp***

#include <iostream>

#include "Object.h"

#include "Person.h"

#include "Student.h"

#include "Vector.h"

using namespace std;

int main() {

system("chcp 1251 > NULL");

Vector v(5);

Person p;

cin >> p;

Student s;

cin >> s;

s.BadMark();

Object\* o = &p;

v.Add(o);

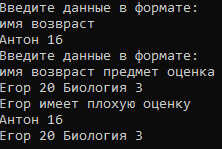
o = &s;

v.Add(o);

cout << v;

return 0;

}



**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;

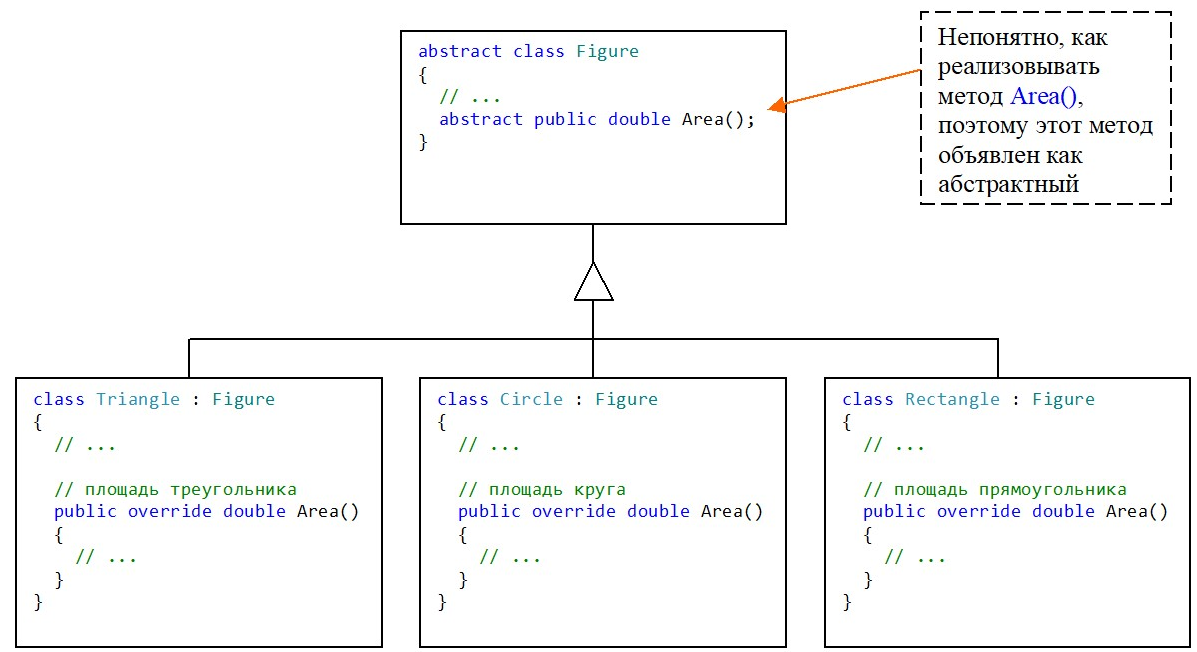
*5. Чем полиморфизм отличается от принципа подстановки?*

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.