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: «Наследование. Полиморфизм. Виртуальные функции»

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Желнин Н.А.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. 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.

Вариант 14

Человек (PERSON)

Имя (name) – string

Возраст (age) – int

Определить методы изменения полей

Создать производный класс TEACHER, имеющий поля Предмет – string и Количество часов - int. Определить методы изменения часов.

Код программы

*#include <iostream>*

*#include "object.h"*

*#include "person.h"*

*#include "teacher.h"*

*#include "vector.h"*

*using namespace std;*

*int main()*

*{*

*Person a;*

*cin>>a;*

*Object \*p=&a;*

*Teacher b;*

*cin>>b;*

*Vector v(5);*

*v.Add(p);*

*p=&b;*

*v.Add(p);*

*cout<<v;*

*}*

*#include "person.h"*

*Person::Person(void){*

*name = "";*

*age = 0;*

*}*

*Person::~Person(void){}*

*Person::Person(string n, int a){*

*name = n;*

*age = a;*

*}*

*Person::Person(const Person& tmp){*

*name = tmp.name;*

*age = tmp.age;*

*}*

*void Person::set\_name(string n){*

*name = n;*

*}*

*void Person::set\_age(int a){*

*age = a;*

*}*

*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 << "\nName:"; in>>p.name;*

*cout << "\nAge:"; in>>p.age;*

*return in;*

*}*

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

*out << "\nName:" <<p.name;*

*out << "\nAge:" <<p.age;*

*return out;*

*}*

*void Person::Show(){*

*cout << "\nName: " << name;*

*cout << "\nAge: " << age;*

*}*

*#pragma once*

*#include "object.h"*

*#include <iostream>*

*#include <string>*

*using namespace std;*

*class Person:*

*public Object*

*{*

*public:*

*Person(void);*

*public:*

*virtual ~Person(void);*

*void Show();*

*Person(string, int);*

*Person(const Person&);*

*string get\_name() {return name;}*

*int get\_age() {return age;}*

*void set\_name(string);*

*void set\_age(int);*

*Person& operator=(const Person&);*

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

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

*protected:*

*string name;*

*int age;*

*};*

*#include "teacher.h"*

*Teacher::Teacher(void):Person() {object = ""; hours = 0;}*

*Teacher::~Teacher(void){}*

*Teacher::Teacher(string n, int a, string o, int h):Person(n, a){*

*object = o;*

*hours = h;*

*}*

*Teacher::Teacher(const Teacher& tmp){*

*name = tmp.name;*

*age = tmp.age;*

*object = tmp.object;*

*hours = tmp.hours;*

*}*

*void Teacher::set\_object(string o){*

*object = o;*

*}*

*void Teacher::set\_hours(int h){*

*hours = h;*

*}*

*void Teacher::Show(){*

*cout << "\nName: " << name;*

*cout << "\nAge: " << age;*

*cout << "\nObject: " << object;*

*cout << "\nHours: " << hours;*

*}*

*void Teacher::change\_hours(char z, int v){*

*if(z == '+') hours += v;*

*if(z == '-') hours -= v;*

*}*

*Teacher& Teacher::operator=(const Teacher&t){*

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

*name=t.name;*

*age=t.age;*

*return \*this;*

*}*

*istream& operator>>(istream&in, Teacher&t){*

*cout << "\nName:"; in>>t.name;*

*cout << "\nAge:"; in>>t.age;*

*cout << "\nObject:"; in>>t.object;*

*cout << "\nHours:"; in>>t.hours;*

*return in;*

*}*

*ostream& operator<<(ostream&out, const Teacher&t){*

*out << "\nName:" <<t.name;*

*out << "\nAge:" <<t.age;*

*out << "\nObject:" <<t.object;*

*out << "\nHours:" <<t.hours;*

*return out;*

*}*

*#pragma once*

*#include "person.h"*

*class Teacher:*

*public Person*

*{*

*public:*

*Teacher(void);*

*public:*

*~Teacher(void);*

*Teacher(string, int, string, int);*

*Teacher(const Teacher&);*

*string get\_object() {return object;}*

*void set\_object(string);*

*int get\_hours() {return hours;}*

*void set\_hours(int);*

*void Show();*

*int hours;*

*void change\_hours(char, int);*

*Teacher& operator=(const Teacher&);*

*friend istream& operator>>(istream&in, Teacher&t);*

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

*protected:*

*string object;*

*};*

*#pragma once*

*class Object*

*{*

*public:*

*//Object(void);*

*//~Object(void);*

*virtual void Show() = 0;*

*};*

*#include "vector.h"*

*Vector::Vector(void){*

*beg=0;*

*size=0;*

*cur=0;*

*}*

*Vector::~Vector(void){*

*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<<"Empty"<<endl;*

*Object \*\*p=v.beg;*

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

*(\*p)->Show();*

*p++;*

*}*

*return out;*

*}*

*#pragma once*

*#include "object.h"*

*#include <string>*

*#include <iostream>*

*using namespace std;*

*class Vector{*

*public:*

*Vector(void);*

*Vector(int);*

*~Vector(void);*

*void Add(Object \*);*

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

*private:*

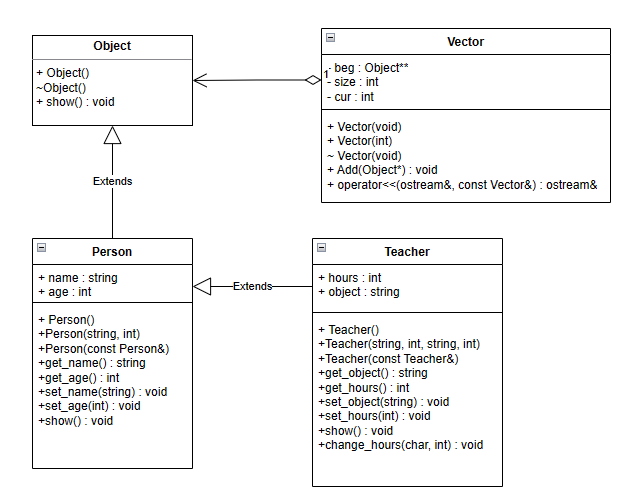
*Object\*\*beg;*

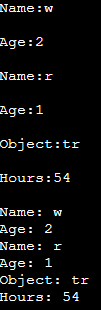
*int size;*

*int cur;*

*};*

Диаграмма и вывод в консоль





Контрольные вопросы

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

Чисто виртуальный метод - виртуальная функция, не имеющая определения в базовом классе. Для страховки от неправильного вызова ее часто объявляют равной нулю: virtual <тип> <имя функции> (<список параметров>) = 0; Чисто виртуальный метод должен переопределяться в производном классе (возможно, опять как чисто виртуальный). Переопределение происходит с помощью спецификатора override (он не обязателен).

2. Какой класс называется абстрактным?

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

3. Для чего предназначены абстрактные классы?

Используется как обобщенная концепция, которая используется для создания конкретных производных классов. Создание объектов абстрактного класса невозможно, однако можно использовать указатели и типы на типы абстрактных классов. Абстрактные классы нельзя использовать для приведения типов, но допускается объявление ссылок и указателей, если абстрактный класс имеет наследников. Если производный класс не определяет все чисто виртуальные функции, он также является абстрактным

Пример:

Нужно сделать 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 : Base {...};

void print(Base\*a){ cout << “Hello!”}

//в основной функции

Derived \*t = new Derived;

print(t); //хотя в функцию передается базовый файл в качестве аргумента.

5. Чем полиморфизм отличается от принципа подстановки?

Полиморфизм - это возможность классов иметь разную реализацию одного и того же функционала в основном с помощью наследования. Это механизм работы языка программирования.

Принцип подстановки является “правилом хорошего кода". В хорошем коде все методы класса-потомка должны быть применимы к объекту класса родителя.

6. Привести примеры иерархий с использованием абстрактных классов.

class Shape {

public:

virtual double getSquare() const = 0; // площадь фигуры

virtual double getPerimeter() const = 0; // периметр фигуры

};

class Rectangle : public Shape { // класс прямоугольника

public:

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

double getSquare() const override {

return width \* height;

}

double getPerimeter() const override {

return width \* 2 + height \* 2;

}

private:

double width; // ширина

double height; // высота

};

class Circle : public Shape { // круг

public:

Circle(double r) : radius(r) { }

double getSquare() const override {

return radius \* radius \* 3.14;

}

double getPerimeter() const override {

return 2 \* 3.14 \* radius;

}

private:

double radius; // радиус круга

};