Министерство образования и науки Российской Федерации

Федеральное государственное образовательное автономное учреждение высшего образования

"Пермский национальный исследовательский политехнический университет"

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

Дисциплина: Основы алгоритмизации и программирования

Тема: Наследование. Виртуальные функции. Полиморфизм.

|  |  |
| --- | --- |
| Выполнил работу | |
| Студент группы РИС-22-1б | |
| Деревнин И.В. | |
|  | |
| Проверил работу | |
| Доцент кафедры ИТАС | |
| Полякова О.А. | |
|  | |

Пермь – 2023

**Анализ предметной области**

**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 основнй функции продемонстрировать перегруженные операции и полиморфизм вектора.

Вариант 15: Базовый класс:

Человек (person)

Имя (name) – string

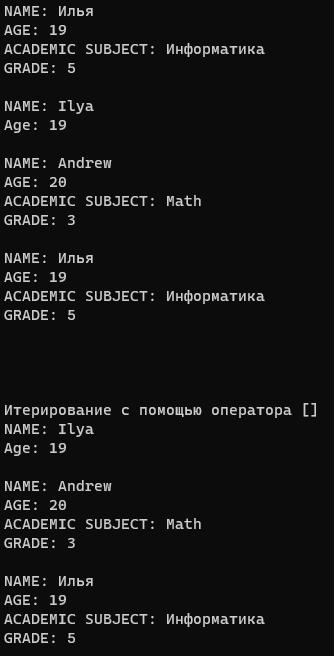
Возраст (age) – int

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.

**Анализ задачи**

Класс Person представляет из себя класс человек, который может существовать отдельно, но имеет очень ограниченный функционал. Класс Person – родитель, от которого можно унаследовать любого человека и добавить к нему недостающие характеристики. Класс student – класс предок, который унаследован от класса Person и расширяет его функционал. У него добавляются 2 поля – предмет и оценка, а также метод, который проверяет удовлетворительна ли оценка. Также определен абстрактный класс object, от которого наследуется остальные классы, исключая вектор. Указатели данного класса могут указывать на любой класс и вызывать методы класса, которые переопределены от виртуальных функций. На этом принципе будет основан класс Вектор.

****Тестирование программы**

*Рис. 1 – Тестирование программы.*

**Заключение**

Была разработана программа, в которой реализован абстрактный класс object и от него унаследованы другие классы – person и student, в которых переопределены абстрактные классы. Для реализации данной программы было описано 4 класса, первый – object, который является родительским абстрактным классом и хранит виртуальную функцию, которые переопределены в классах наследниках. Второй – класс person, который является наследником класса object, расширяет его функционал и переопределяет чисто виртуальную функцию. Третий класс – student, который является наследником классов person и object, он расширяет их функционал и переопределяет виртуальную функцию. Четвертый класс – класс вектор, который представляет из себя динамический массив указателей object, через данные указатели можно обращаться к полям классов наследников и также хранить их в динамической памяти.

# Ответы на 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;

}

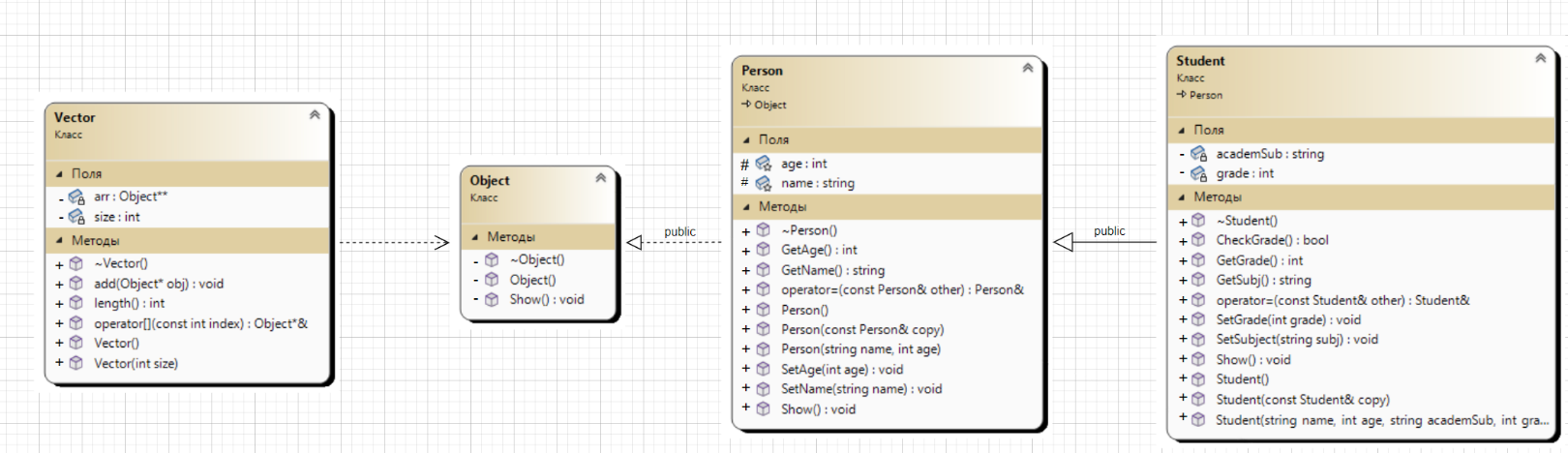
private:

double radius; // радиус круга

};

**Приложения**

Приложение UML-диаграмма



Приложение Б – код программы

Main.cpp:

#include "Student.h"

#include "Vector.h"

int main()

{

system("chcp 1251 >> null");

Person pers("Ilya", 19);

Student stud("Andrew", 20, "Math", 3), stud2;

cin >> stud2;

Vector v(2);

Object\* a = &pers; v.add(a);

a = &stud; v.add(a);

a = &stud2; v.add(a);

cout << endl;

cout << v;

cout << endl << endl << endl;

cout << "Итерирование с помощью оператора []" << endl;

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

{

v[i]->Show(); cout << endl;

}

return 0;

}

person.h:

#pragma once

#include "Object.h"

class Person : public Object

{

private:

friend ostream& operator <<(ostream& os, const Person& per);

friend istream& operator >> (istream& is, Person& per);

protected:

string name; //наследуемые поля класса

int age;

public:

Person(); //конструкторы / деструкторы

Person(string name, int age);

Person(const Person& copy);

~Person();

void Show() override; //переопрделенный метод абстрактного класса

void SetName(string name) { this->name = name; } //сеттеры / геттеры

void SetAge(int age) { this->age = age; }

string GetName() { return this->name; }

int GetAge() { return this->age; }

Person& operator =(const Person& other); //оператор примсваивания

};

person.cpp:

#include "Person.h"

Person::Person()

{

cout << "Конструктор без параметров Person" << endl;

name = "TEMP";

age = 0;

}

Person::Person(string name, int age)

{

cout << "Конструктор с параметрами Person" << endl;

this->name = name;

this->age = age;

}

Person::Person(const Person& copy)

{

cout << "Конструктор копирования Person" << endl;

this->name = copy.name;

this->age = copy.age;

}

ostream& operator<<(ostream& os, const Person& per)

{

os << "Name: " << per.name << "\nAge: " << per.age;

return os;

}

istream& operator>>(istream& is, Person& per)

{

is >> per.name >> per.age;

return is;

}

Person& Person::operator=(const Person& other)

{

this->name = other.name;

this->age = other.age;

return \*this;

}

Person::~Person()

{

cout << "Деструктор класса Person" << endl;

age = -1;

name = "";

}

void Person::Show()

{

cout << "NAME: " << name << "\nAge: " << age << endl;

}

Student.h:

#pragma once

#include "Person.h"

class Student : public Person //Наследование от класса person

{

private:

string academSub; //дополнительные поля класса

int grade;

friend istream& operator >> (istream& is, Student& per); //операторы ввода/вывода

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

public:

Student(); //конструкторы / деструкторы

Student(string name, int age, string academSub, int grade);

Student(const Student& copy);

~Student();

void Show() override; //переопределенный метод

void SetSubject(string subj) { this->academSub = subj; } //сеттеры / геттеры

void SetGrade(int grade) { this->grade = grade; }

string GetSubj() { return this->academSub; }

int GetGrade() { return this->grade; }

bool CheckGrade();

Student& operator =(const Student& other); //оператор присваиваняия

};

Student.cpp:

#include "Student.h"

Student::Student() :Person()

{

cout << "Конструктор без параметров класса Student" << endl;

academSub = "TEMP";

grade = 0;

}

Student::Student(string name, int age, string academSub, int grade) :Person(name, age)

{

cout << "Конструктор c параметрами класса Student" << endl;

this->academSub = academSub;

this->grade = grade;

}

Student::Student(const Student& copy) :Person(copy)

{

cout << "Конструктор копирования класса Student" << endl;

this->academSub = copy.academSub;

this->grade = copy.grade;

}

ostream& operator<<(ostream& os, const Student& st)

{

os << "Name: " << st.name << "\nAge: " << st.age << "\nAcademic subject: " << st.academSub << "\nGrade: " << st.grade;

return os;

}

istream& operator>>(istream& is, Student& per)

{

cout << "NAME: ";is >> per.name;

cout << "AGE: ";is >> per.age;

cout << "ACADEMIC SUBJECT: ";is >> per.academSub;

cout << "GRADE: "; is >> per.grade;

return is;

}

Student& Student::operator=(const Student& other)

{

this->academSub = other.academSub;

this->age = other.age;

this->grade = other.grade;

this->name = other.name;

return \*this;

}

Student::~Student()

{

cout << "Деструктор класса Student" << endl;

grade = 0;

academSub = "";

}

bool Student::CheckGrade()

{

if (this->grade <=2)

{

return false;

}

return true;

}

void Student::Show()

{

cout << "NAME: " << name << "\nAGE: " << age << "\nACADEMIC SUBJECT: " << academSub << "\nGRADE: " << grade << endl;

}

Vector.h:

#pragma once

#include "Object.h"

class Vector

{

private:

Object\*\* arr; //указатель на указатель, для того, чтобы хранить любые данные

int size; //дллина вектора

friend ostream& operator <<(ostream& os, const Vector& vec); //оператор вывода всего вектора в консоль

public:

int length() { return this->size; } //Метод возвращаюзий длину вектора

Vector(); //конструкторы / деструкторы

Vector(int size);

~Vector();

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

Object\*& operator[](const int index); //оператор индексирования по вектору

};

Vector.cpp:

#include "Vector.h"

Vector::Vector()

{

arr = nullptr;

size = 0;

}

Vector::Vector(int size)

{

this->size = size;

this->arr = new Object\* [size];

for (int i = 0; i < size; i++)

arr[i] = nullptr;

}

Vector::~Vector()

{

delete[]arr;

}

void Vector::add(Object\* obj)

{

int i = 0;

while (arr[i] != nullptr && i < size) ++i;

if (i < size) arr[i] = obj;

else

{

Object\*\* temp = new Object\* [size + 1];

for (int j = 0; j < size; j++)

temp[j] = arr[j];

temp[size] = obj;

++size;

delete[]arr;

arr = temp;

temp = nullptr;

}

}

Object\*& Vector::operator[](const int index)

{

return arr[index];

}

ostream& operator<<(ostream& os, const Vector& vec)

{

for (int i = 0; i < vec.size && vec.arr[i] != nullptr; ++i)

{

vec.arr[i]->Show();

cout << endl;

}

return os;

}