МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра информационные технологии и автоматизированные системы

**Дисциплина Информатика**

**Полиморфизм**

Выполнил студент ИВТ-22-2б:

Мифтахов Марат Ринатович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Пермь, 2023

**Постановка задачи**

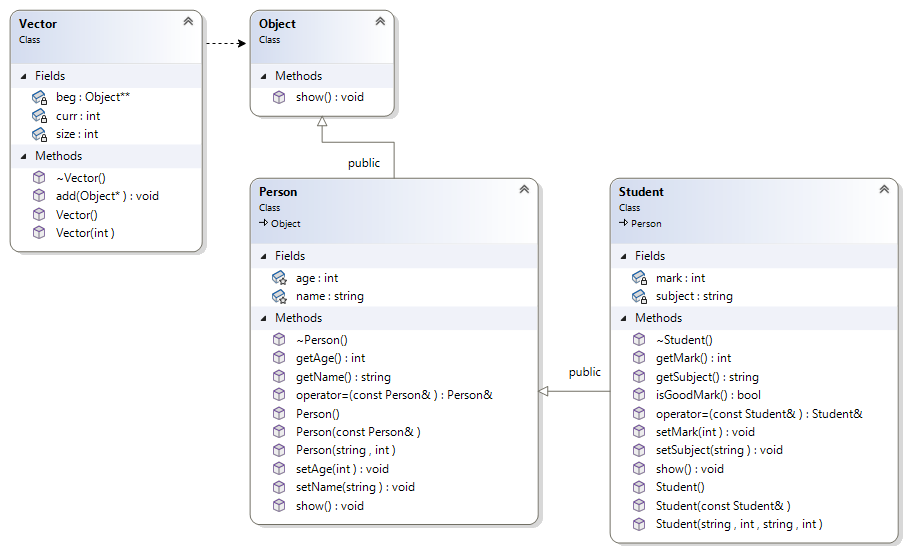
1. Определить абстрактный класс.
2. Определить иерархию классов, в основе которой будет находиться абстрактный класс (см. лабораторную работу №4).
3. Определить класс Вектор, элементами которого будут указатели на объекты иерархии классов.
4. Перегрузить для класса Вектор операцию вывода объектов с помощью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

**Задание**

1. Класс Person
2. Поля имя — string, возраст — int
3. Определить геттеры и сеттеры
4. Создать производный класс Student
5. Поля предмет — string, возраст — int
6. Определить метод изменения полей, выдающий сообщение о неудовлетворительной оценке

**Диаграмма классов**

Ниже представлена UML-диаграмма классов.



**Программный код**

Файл Vector.cpp:

#include "Vector.h"

Vector::Vector() {

beg = 0;

size = 0;

curr = 0;

}

Vector::Vector(int size) {

this->size = size;

curr = 0;

beg = new Object \* [size];

}

Vector::~Vector() {

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

beg = 0;

}

void Vector::add(Object\* someObj) {

if (curr < size) {

beg[curr] = someObj;

curr++;

}

}

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

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

Object\*\* p = vect.beg;

// суть в вызове метода show некоторого объекта

for (int i = 0; i < vect.curr; i++) {

(\*p)->show();

p++;

}

return out;

}

Файл Person.cpp:

#include "Person.h"

#include "Student.h"

///--------------------Person----------------///

Person::Person() {

name = "";

age = 0;

}

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

this->name = name;

this->age = age;

}

Person::Person(const Person& p) {

name = p.name;

age = p.age;

}

Person::~Person() {}

void Person::setName(string name) {

this->name = name;

}

string Person::getName() {

return name;

}

void Person::setAge(int age) {

this->age = age;

}

int Person::getAge() {

return age;

}

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

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

name = p.name;

age = p.age;

return \*this;

}

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

return out << "name: " << p.name << " age: " << p.age;

}

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

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

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

return in;

}

void Person::show() {

cout << "name: " << name << endl;

cout << "age: " << age << endl;

}

///--------------------Person----------------///

Файл Student.cpp:

#include "Person.h"

#include "Student.h"

///--------------------Student----------------///

Student::Student() : Person::Person() {

subject = "";

mark = 0;

}

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

this->subject = subject;

this->mark = mark;

}

Student::Student(const Student& p) {

name = p.name;

age = p.age;

subject = p.subject;

mark = p.mark;

}

Student::~Student() {}

void Student::setSubject(string subject) {

this->subject = subject;

}

string Student::getSubject() {

return subject;

}

void Student::setMark(int mark) {

this->mark = mark;

}

int Student::getMark() {

return mark;

}

bool Student::isGoodMark() {

if (mark > 3) return true;

else return false;

}

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

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

this->name = ptr.name;

this->age = ptr.age;

this->subject = ptr.subject;

this->mark = ptr.mark;

return \*this;

}

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

cout << "name:? " << endl; in >> ptr.name;

cout << "age:?" << endl; in >> ptr.age;

cout << "subject:?" << endl; in >> ptr.subject;

cout << "mark:?" << endl; in >> ptr.mark;

return in;

}

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

return out << "name: " << p.name << " age: " << p.age << "\n" <<

"subject: " << p.subject << " mark: " << p.mark << endl;

}

///--------------------Student----------------///

Файл Vector.h:

#include "Person.h"

#include "Student.h"

#pragma once

class Vector

{

int size;

int curr;

Object\*\* beg;

public:

Vector();

Vector(int);

~Vector();

void add(Object\*);

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

};

Файл Person.h:

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Object {

public:

virtual void show() = 0;

};

class Person : public Object {

protected:

string name;

int age;

public:

Person();

Person(string, int);

Person(const Person&);

virtual ~Person(); //объявление виртуальным для гарантии выполнения деструктора производного класса

void setName(string);

string getName();

void setAge(int);

int getAge();

Person& operator=(const Person&);

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

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

void show() override;

};

Файл Student.h:

#pragma once

#include "Person.h"

class Student : public Person {

string subject;

int mark;

public:

Student();

Student(string, int, string, int);

Student(const Student&);

~Student();

void setSubject(string);

string getSubject();

void setMark(int);

int getMark();

bool isGoodMark();

Student& operator=(const Student&);

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

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

void show() {

Person::show();

cout << "subj: " << subject << endl;

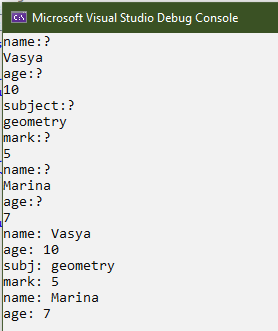
cout << "mark: " << mark << endl;

}

};

**Вывод программы**

Ниже представлен вывод программы на консоль.

****

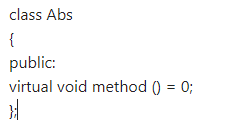
**Контрольные вопросы**

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

Ответ: чисто виртуальный метод — метод, не имеющий определения в базовом классе. Отличается от виртуального отсутствием тела.

1. Какой класс называется абстрактным?

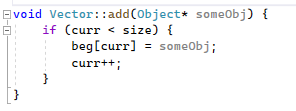
Ответ: абстрактный класс — класс, который содержит или наследует хотя бы одну чисто виртуальную функцию без определения.

1. Для чего предназначены абстрактные классы?

Ответ: для создания конкретных производных классов.

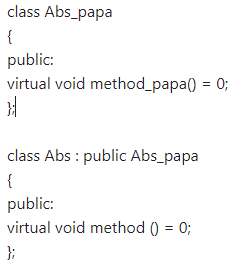
1. Что такое полиморфные функции?

Ответ: полиморфные функции — функции, имеющие экземпляр базового класса в качестве параметра и принимающие экземпляры производных классов.



1. Чем полиморфизм отличается от принципа подстановки?

Ответ: принцип подстановки — принцип полиморфизма

1. Привести примеры иерархий с использованием абстрактных классов.
2. Привести примеры полиморфных функций.

Ответ: функция add вектора (см. п. 4)

1. В каких случаях используется механизм позднего связывания?

Ответ: В случаях вызова полиморфной функции с типом параметра класса-наследника