Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

Лабораторная работа №5

«Наследование. Виртуальные функции. Полиморфизм»

Выполнил:

студент первого курса

ЭТФ группы РИС-23-3б

Коротаев Александр Дмитриевич

Проверила:

Доцент кафедры ИТАС О. А. Полякова

Пермь, 2024

**Цель задания**

1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2. Создание иерархии классов с использованием простого наследования и абстрактного класса.
3. Изучение полиморфизма и виртуальных методов.

**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.

**Задание**

Базовый класс:

ЧЕЛОВЕК (PERSON)

Имя (name) – string Возраст (age) – int

Определить методы изменения полей.

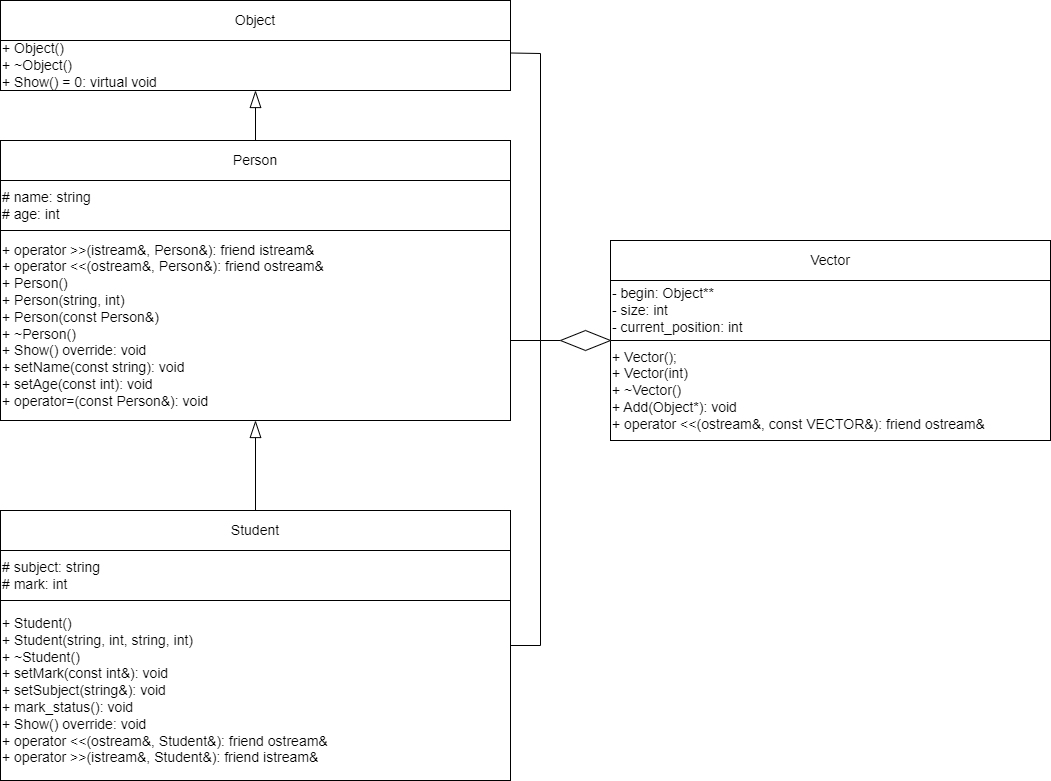
Создать производный класс STUDENT, имеющий поля Предмет – string

* Оценка – int. Определить методы изменения полей и метод, выдающий сообщение о неудовлетворительной оценке.

**Анализ задачи**

1. Необходимо реализовать класс Person и дочерний для него класс
2. Определить класс Вектор, элементами которого будут указатели на объекты иерархии классов.
3. Класс Person содержит защищенные данные name и age, что предотвращает их изменение извне класса. Также определены конструктор по умолчанию, конструктор с параметрами, конструктор копирования и деструктор. Предоставлены функции для установки значений и вывода информации о персоне. Определён оператор присваивания и дружественные функции для работы с потоками ввода/вывода.
4. Класс Student является производным от Person. Он добавляет свои собственные защищенные данные subject и mark, а также предоставляет дополнительные функции для установки этих значений и вывода информации
5. студенте. Определены конструктор с параметрами, функции для установки предмета и оценки, а также дружественные функции для работы с потоками ввода/вывода.
6. Перегруженные операторы +, -, = и << позволяют выполнять различные операции с объектами класса. Например, оператор + позволяет складывать целое число с первым полем объекта класса и вещественное число со вторым полем соответственно. Оператор << позволяет выводить объекты класса в поток вывода.
7. В главной функции показаны операции, которые можно выполнить с объектом классов.

**Блок схема**

****

**Код**

Файл ООП5.cpp

#include <iostream>

#include <string>

using namespace std;

#include "Person.h"

#include "VECTOR.h"

int main()

{

system("chcp 1251>null");

Person person1 ("faokfaf", 12);

cout << person1;

person1.Show();

Student student1("Вася", 34, "инфа", 3);

cout << student1;

student1.Show();

Vector vector(5);

Person person2("Максим", 19);

cin >> person2;

Student student2;

cin >> student2;

Object\* ptr = &person2;

vector.Add(ptr);

ptr = &student2;

vector.Add(ptr);

cout << endl;

cout << vector;

return 0;

}

Файл Object.h

#pragma once

class Object

{

public:

Object(){}

~Object(){}

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

};

Файл Person.h

#pragma once

#include "Object.h"

class Person : public Object

{

protected:

string name;

int age;

public:

Person();

Person(string, int);

Person(const Person&);

~Person();

void Show() override;

void setName(const string);

void setAge(const int);

void operator = (const Person&);

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

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

};

class Student : public Person

{

protected:

string subject;

int mark;

public:

Student();

Student(string, int, string, int);

void setSubject(string);

void setMark(int);

void mark\_status();

void Show() override;

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

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

};

Файл Person.cpp

#pragma once

#include <iostream>

#include <string>

using namespace std;

#include "Person.h"

Person::Person()

{

cout << "Вызвался конструктор без параметров для объекта " << this << endl;

this->name = "никто";

this->age = 0;

}

Person::Person(string name, int age)

{

cout << "Вызвался конструктор с параметрами для объекта " << this << endl;

this->name = name;

this->age = age;

}

Person::Person(const Person& other)

{

cout << "Вызвался конструктор копирования для объекта " << this << endl;

this->name = other.name;

this->age = other.age;

}

Person::~Person()

{

cout << "Вызвался деструктор для объекта " << this << endl;

}

void Person::Show()

{

cout << "ФИО: " << this->name << "\nВозраст: " << this->age << "\n\n";

}

void Person::setName(const string name)

{

this->name = name;

}

void Person::setAge(const int age)

{

this->age = age;

}

void Person::operator=(const Person& person)

{

this->name = person.name;

this->age = person.age;

}

ostream& operator<<(ostream& stream, Person& person)

{

stream << "ФИО: " << person.name << "\nВозраст: " << person.age << "\n\n";

return stream;

}

istream& operator>>(istream& stream, Person& person)

{

string name; int age;

cout << "Введите имя: ";

cin.seekg(cin.eof());

getline(cin, name);

cout << "Введите возраст: ";

cin >> age;

person.setName(name);

person.setAge(age);

return stream;

}

Student::Student()

{

this->name = "Никто Никтотович Никтотович";

this->age = 0;

this->subject = "Обед";

this->mark = 5;

}

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

{

this->name = name;

this->age = age;

this->subject = subject;

this->mark = mark;

}

void Student::setSubject(string subject)

{

this->subject = subject;

}

void Student::setMark(int mark)

{

this->mark = mark;

}

void Student::mark\_status()

{

if (this->mark < 3)

{

cout << "Ты отчислен =(\n\n";

}

else

{

cout << "Все хорошо, пока учись =)\n\n";

}

}

void Student::Show()

{

cout << "Имя: " << this->name << "\nВозраст: " << this->age << endl;

cout << "Предмет: " << this->subject << "\nОценка: " << this->mark << "\n\n";

}

ostream& operator<<(ostream& stream, Student& student)

{

stream << "Имя: " << student.name << endl << "Возраст: " << student.age << endl;

stream << "Предмет: " << student.subject << endl << "Оценка: " << student.mark << "\n\n";

return stream;

}

istream& operator >>(istream& stream, Student& student)

{

string subject; int mark;

cout << "Введите предмет: ";

cin.seekg(cin.eof());

getline(cin, subject);

cout << "Введите оценку: ";

stream >> mark;

student.setSubject(subject);

student.setMark(mark);

return stream;

}

Файл VCTOR.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

#include "Object.h"

class Vector

{

private:

Object\*\* begin;

int size, current\_position;

public:

Vector();

Vector(int);

~Vector();

void Add(Object\*);

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

};

Файл VECTOR.cpp

#pragma once

#include "VECTOR.h"

Vector::Vector()

{

this->begin = 0;

this->size = 0;

this->current\_position = 0;

}

Vector::Vector(int size)

{

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

this->size = size;

this->current\_position = 0;

}

Vector::~Vector()

{

if (begin != 0)

{

delete[] begin;

}

begin = 0;

}

void Vector::Add(Object\* ptr)

{

if (current\_position < size)

{

begin[current\_position] = ptr;

current\_position++;

}

}

ostream& operator<<(ostream& stream, const Vector& vector)

{

if (vector.size == 0)

{

cout << "Пусто =(\n";

}

else

{

Object\*\* ptr = vector.begin;

for (int i = 0; i < vector.current\_position; i++)

{

(\*ptr)->Show();

ptr++;

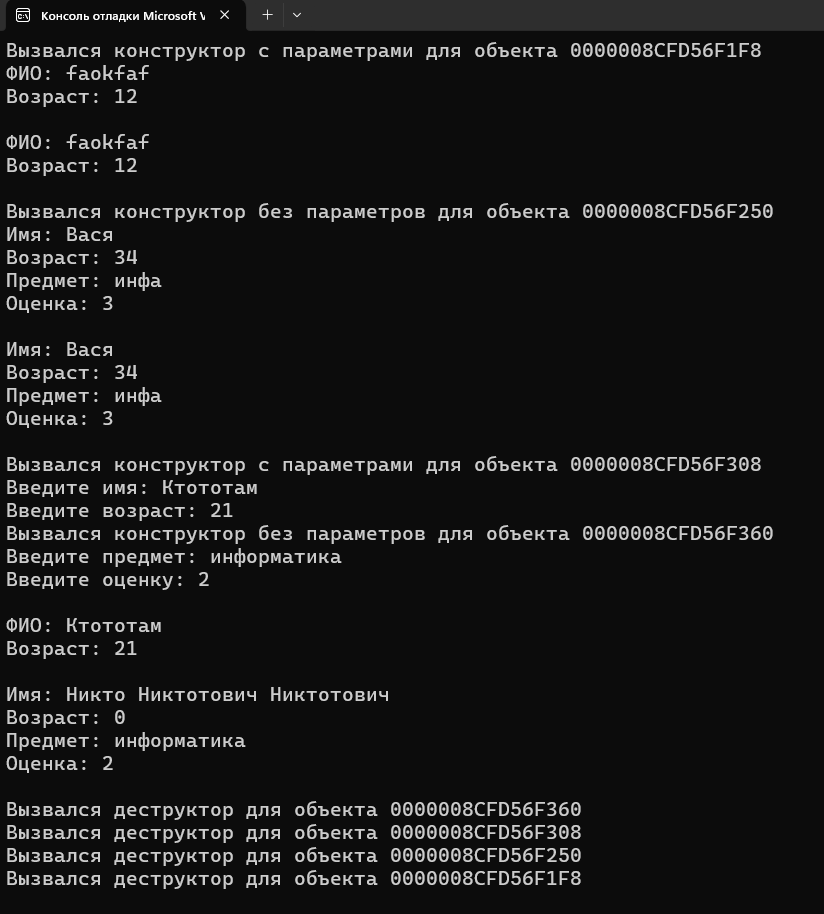
}

}

return stream;

}

**Результаты работы**



**GitHub**

<https://github.com/Korovay4ik/Laboratory-works>

**Контрольные вопросы**

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

Чисто виртуальный метод (также известный как pure virtual method) - это метод, который объявлен в базовом классе, но его реализация не предоставлена. Вместо этого, он имеет только декларацию (т.е. прототип функции), и в конце этой декларации стоит ключевое слово = 0;.

<тип> <имя функции> (<список параметров>) = 0;

Это означает, что производные классы обязаны реализовать этот метод, иначе они сами станут абстрактными и не могут быть инстанцированы. Виртуальный метод, в отличие от чисто виртуального, может иметь реализацию в базовом классе, которая может быть переопределена в производном классе.

Основное различие между виртуальной функцией и чисто виртуальной функцией является то, что виртуальная функция является функцией в базовом классе, который объявлен с использованием виртуального ключевого слова в то время, как чистая виртуальная функция является виртуальной функцией в базовом классе без определения функции.

1. **Какой класс называется абстрактным?**

Абстрактный класс - это класс, который содержит хотя бы один чисто виртуальный метод. Такой класс сам по себе не может быть инстанцирован, так как не предоставляет полную реализацию всех методов. Он служит в качестве основы для определения интерфейса для других классов.

1. **Для чего предназначены абстрактные классы?**

Абстрактные классы предназначены для представления общих понятий, которые предполагается конкретизировать в производных классах. Абстрактный класс может использоваться только в качестве базового для других классов — объекты абстрактного класса создавать нельзя, поскольку прямой или косвенный вызов чисто виртуального метода приводит к ошибке при выполнении.

1. **Что такое полиморфные функции?**

Полиморфные функции - это функции, которые могут быть вызваны с различными типами аргументов, благодаря механизму позднего связывания. Это позволяет использовать одну и ту же функцию с разными объектами, даже если они принадлежат к разным классам. Понятие позднее связывание означает, что код вызова нужной функции фор-мируется при выполнении программы.

1. **Чем полиморфизм отличается от принципа подстановки?**

При полиморфизме функция может изменять свой функционал. Полиморфизм - это способность объекта одного класса вести себя как

объект другого класса. Это достигается через наследование и виртуальные функции. Принцип подстановления говорит о том, что объекты можно заменять объектами их типов без изменения программы. Полиморфизм идет дальше, позволяя объектам разных классов заменяться друг другом в определенных контекстах.

1. **Привести примеры иерархий с использованием абстрактных**

**классов.**

#include <iostream>

* Абстрактный класс Animal class Animal {

public:

virtual void makeSound() = 0; // Чисто виртуальная функция, делает класс абстрактным

};

* Класс Dog, наследуется от Animal

class Dog : public Animal {

public:

void makeSound() override {

std::cout << "Woof" << std::endl;

}

};

* Класс Cat, наследуется от Animal class Cat : public Animal {

public:

void makeSound() override { std::cout << "Meow" << std::endl;

}

};

int main() {

Dog dog;

Cat cat;

dog.makeSound(); // Выводит "Woof"

cat.makeSound(); // Выводит "Meow"

return 0;

}

1. **Привести примеры полиморфных функций.**

class Abstract { //Абстрактный класс

public:virtual void print\_msg() = 0;

};

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

Позднее связывание означает, что объект связывается с вызовом функции только во время исполнения программы, а не раньше. Позднее связывание достигается в С++ с помощью использования виртуальных функций и производных классов.

Механизм позднего связывания используется в следующих случаях:

* Когда вызывается виртуальный метод объекта.
* Когда происходит вызов функции через указатель или ссылку на базовый класс, где фактический вызов осуществляется для производного класса.
* В шаблонах, когда конкретный тип определяется во время компиляции.