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

ЛАБОРАТОРНАЯ РАБОТА

НАСЛЕДОВАНИЕ. ВИРТУАЛЬНЫЕ ФУНКЦИИ. ПОЛИМОРФИЗМ.

Выполнил: студент группы РИС-23-3б

Артем Владимирович Швецов

Проверила: доцент кафедры ИТАС

Ольга Андреевна Полякова

Пермь 2024

**Постановка задачи**

1. Определить абстрактный класс.
2. Определить иерархию классов, в основе абстрактный класс.
3. Определить класс Вектор, элементы – указатели на объекты иерархии.
4. Перегрузить для класса Вектор операцию вывода с помощью потоков.

Для варианта 15:

Базовый класс:

ЧЕЛОВЕК (PERSON)

Имя – string

Возраст – int

Определить методы изменения полей

Производный класс:

STUDENT

Предмет – string

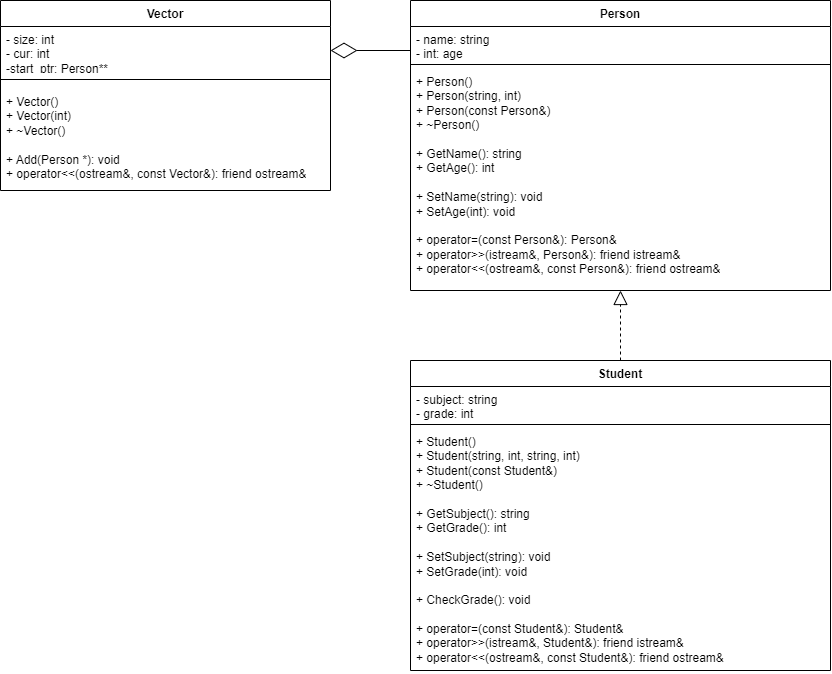
Оценка – int

Определить методы изменения полей и метод сообщающий о неудовлетворительной оценке.

**Анализ задачи**

1. Создается новый файл с расширением .h, где определяется новый класс Person.
2. Создается парный .cpp файл, в котором определяются методы нового класса.
3. Создается дополнительная пара файлов, описывающая класс Vector и его методы.
4. Поле класса vector - указатель на первый элемент списка.

**UML диаграмма**



**Код**

Person.h

#pragma once

#include <string>

using namespace std;

class LivingObject

{

protected:

virtual bool IsExist() = 0;

virtual void Show() = 0;

};

class Person : LivingObject {

protected:

string name;

int age;

public:

Person();

Person(string, int);

Person(const Person&);

~Person();

string GetName();

int GetAge();

void SetName(string);

void SetAge(int);

bool IsExist() override;

virtual void Show() override;

Person& operator=(const Person&);

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

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

};

class Student: public Person {

protected:

string subject;

int grade;

public:

Student();

Student(string, int, string, int);

Student(const Student&);

~Student();

string GetSubject();

int GetGrade();

void SetSubject(string);

void SetGrade(int);

void CheckGrade();

virtual void Show() override;

Student& operator=(const Student&);

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

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

};

Person.cpp

#pragma once

#include "Person.h"

#include <string>

#include <iostream>

using namespace std;

Person::Person()

{

name = "noname";

age = 0;

}

Person::Person(string inName, int inAge)

{

name = inName;

age = inAge;

}

Person::Person(const Person& p)

{

name = p.name;

age = p.age;

}

Person::~Person()

{

}

string Person::GetName()

{

return name;

}

int Person::GetAge()

{

return age;

}

void Person::SetName(string inName)

{

name = inName;

}

void Person::SetAge(int inAge)

{

age = inAge;

}

bool Person::IsExist() {

if (name != "noname" && age != 0)

{

return true;

}

else

{

return false;

}

}

void Person::Show() {

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

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

}

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

{

this->name = p.name;

this->age = p.age;

return \*this;

}

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

{

cout << "Set name: ";

stream >> p.name;

cout << "Set age: ";

stream >> p.age;

return stream;

}

ostream& operator<<(ostream& stream, const Person& p)

{

stream << "Name: " << p.name << endl;

stream << "Age: " << p.age << endl;

return stream;

}

Student::Student():Person()

{

subject = "nosubject";

grade = 0;

}

Student::Student(string inName, int inAge, string inSubject = "nosubject", int inGrade = 0) : Person(inName, inAge)

{

subject = inSubject;

grade = inGrade;

}

Student::Student(const Student& s)

{

name = s.name;

age = s.age;

subject = s.subject;

grade = s.grade;

}

Student::~Student()

{

Person::~Person();

}

string Student::GetSubject()

{

return subject;

}

int Student::GetGrade()

{

return grade;

}

void Student::SetSubject(string inSubject)

{

subject = inSubject;

}

void Student::SetGrade(int inGrade)

{

grade = inGrade;

}

void Student::CheckGrade()

{

if (grade < 3)

{

cout << "Grade is not satisfactory" << endl;

}

}

void Student::Show() {

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

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

cout << "Subject: " << subject << endl;

cout << "Grade: " << grade << endl;

}

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

{

this->name = s.name;

this->age = s.age;

this->subject = s.subject;

this->grade = s.grade;

return \*this;

}

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

{

cout << "Set name: ";

stream >> s.name;

cout << "Set age: ";

stream >> s.age;

cout << "Set subject: ";

stream >> s.subject;

cout << "Set grade: ";

stream >> s.grade;

return stream;

}

ostream& operator<<(ostream& stream, const Student& s)

{

stream << "Name: " << s.name << endl;

stream << "Age: " << s.age << endl;

stream << "Subject: " << s.subject << endl;

stream << "Grade: " << s.grade << endl;

return stream;

}

Vector.h

#pragma once

#include "Person.h"

class Vector {

int size;

int cur;

Person\*\* start\_ptr;

public:

Vector();

Vector(int);

~Vector();

void Add(Person\*);

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

};

Vector.cpp

#pragma once

#include "Vector.h"

#include <string>

#include <iostream>

Vector::Vector()

{

start\_ptr = nullptr;

size = 0;

cur = 0;

}

Vector::Vector(int n)

{

start\_ptr = new Person\* [n];

size = n;

cur = 0;

}

Vector::~Vector() {

if (start\_ptr != nullptr)delete[] start\_ptr;

start\_ptr = nullptr;

}

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

if (cur < size)

{

start\_ptr[cur] = p;

cur++;

}

else

{

cout << "Vector overflowed" << endl;

}

}

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

{

if (v.size == 0) {

stream << "Empty vector" << endl;

}

Person\*\* p = v.start\_ptr;

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

{

//stream << \*\*p;

(\*\*p).Show();

p++;

}

return stream;

}

Main.cpp

#include "Person.h"

#include "Vector.h"

#include <string>

#include <iostream>

using namespace std;

int main() {

Vector vect(2);

Person p1;

Student s1;

p1.SetName("ABC");

p1.SetAge(90);

s1.SetName("ABC jr.");

s1.SetAge(30);

s1.SetSubject("Inf");

s1.SetGrade(5);

vect.Add(&p1);

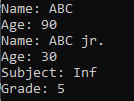
vect.Add(&s1);

cout << vect;

return 0;

}

**Решение**



**Выводы**

В ходе выполнения работы были изучены принципы наследования и полиморфизма, а также абстрактные классы и методы работы с ними.

**Ответы на контрольные вопросы:**

1. Чисто виртуальный метод – виртуальный метод у которого нет определения (=0).
2. Класс, который содержит хотя бы одну чисто виртуальную функцию.
3. Предназначены для создания иерархии классов с общим интерфейсом.
4. Функции, меняющие выполняемые действия в зависимости от класса.
5. При постановке действия не меняются, при полиморфизме – меняются в зависимости от класса.

class Object {

virtual void Show() = 0;

}

class Cube : public Object{

virtual void Show() {

cout << “This is a cube”;

}

}

void save (int i, Object\* objj){

vect[i] = objj;

}

**Github**

<https://github.com/Hitikov/Lab_OOP_5>