Министерство образования и науки Российской Федерации

Федеральное государственное образовательное автономное учреждение высшего образования

"Пермский национальный исследовательский политехнический университет"

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

Дисциплина: Основы алгоритмизации и программирования

Тема: Простое наследование. Принцип подстановки.

|  |  |
| --- | --- |
| 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 пользовательский класс.
2. Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.
3. Определить в классе деструктор.
4. Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы).
5. Перегрузить операции: присваивания, вывода и ввода объектов с помощью потоков.
6. Определить производный класс.
7. Написать демонстрационную программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.
8. Реализовать функции, получающие и возвращающие объект базового класса. Продемонстрировать принцип подстановки.

Вариант 15: Базовый класс:

Человек (person)

Имя (name) – string

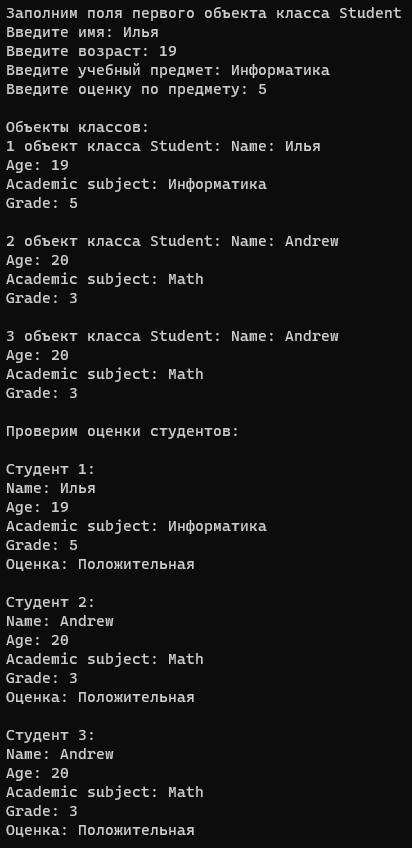
Возраст (age) – int

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.

****Тестирование программы**

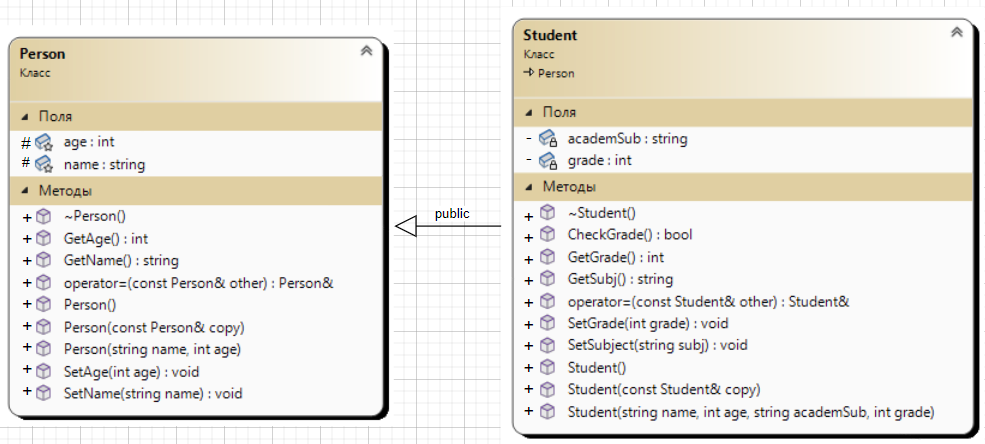
*Рис. 1 – Тестирование программы и классов student и person.*

**Заключение**

Была разработана программа, которая с помощью наследования классов расширяет возможности класса родителя. Для реализации данной программы было описано 2 класса, первый – person, который является родительским классом и хранит стандартные методы и поля. И второй – класс student, который является наследником класса person, и расширяет его функционал, а также количество полей.

**Приложения**

Приложение UML-диаграмма



Приложение Б – код программы

Main.cpp:

#include "Student.h"

int main()

{

system("chcp 1251 >> null");

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

Person pers;

cout << "\nСоздание Person с параметрами" << endl;

Person pers2("Ilya", 19);

cout << "\nСоздание Student без параметров" << endl;

Student st;

cout << "\nСоздание Student с параметрами" << endl;

Student st2("Andrew", 20, "Math", 3);

cout << "\nКопирование предудущего объекта в новый." << endl;

Student st3(st2);

cout << "\nЗаполним поля первого объекта класса Student" << endl;

string str;

int a;

cout << "Введите имя: "; cin >> str;

cout << "Введите возраст: "; cin >> a;

st.SetAge(a); st.SetName(str);

cout << "Введите учебный предмет: "; cin >> str;

cout << "Введите оценку по предмету: "; cin >> a;

st.SetGrade(a); st.SetSubject(str);

cout << endl << "Объекты классов: " << endl;

cout << "1 oбъект класса Student: " << st << endl << endl;

cout << "2 oбъект класса Student: " << st2 << endl << endl;

cout << "3 oбъект класса Student: " << st3 << endl << endl;

cout << "Проверим оценки студентов: " << endl << endl;

if (st.CheckGrade()) str = "Положительная";

else str = "Отрицательная";

cout << "Студент 1: " << endl << st << endl << "Оценка: " << str << endl << endl;

if (st2.CheckGrade()) str = "Положительная";

else str = "Отрицательная";

cout << "Студент 2: " << endl << st2 << endl << "Оценка: " << str << endl << endl;

if (st3.CheckGrade()) str = "Положительная";

else str = "Отрицательная";

cout << "Студент 3: " << endl << st3 << endl << "Оценка: " << str << endl << endl;

cout << endl << "Обратим внимание, что деструкторы вызываются обратно конструкторам." << endl;

return 0;

}

person.h:

#pragma once

#include <iostream>

using namespace std;

class Person //класс родитель

{

private:

friend ostream& operator <<(ostream& os, const Person& per); //операторы ввода/вывода

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

protected:

string name; //поля класса, которые будут унаследованы

int age;

public:

Person(); //конструкторы и деструкторы

Person(string name, int age);

Person(const Person& copy);

~Person();

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

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

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

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

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

};

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 = "";

}

Student.h:

#pragma once

#include "Person.h"

class Student : public Person //дочерний класс от класса родителя, наследование

{

private:

string academSub; //доп поля класса, относительно класса родителя

int grade;

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

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

public:

Student(); //конструкторы и деструктор

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

Student(const Student& copy);

~Student();

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

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

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

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

bool CheckGrade(); //проверка оценки на положительная/отрицательная

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

};

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 с 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)

{

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

}