Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

# Лабораторная работа №4

# « Простое наследование. Принцип подстановки »

# Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Юхновец В.Г.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

**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 операцию присваивания.

6. Перегрузить операции ввода и вывода объектов с помощью потоков.

7. Определить производный класс.

8. Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.

9. 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.

**Ответы на контрольные вопросы**

1. **Для чего используется механизм наследования?** Механизм наследования используется в объектно-ориентированном программировании для создания новых классов на основе уже существующих классов. При наследовании производный класс получает доступ к компонентам базового класса и может добавлять свои собственные компоненты и функциональность.
2. **Каким образом наследуются компоненты класса, описанные со спецификатором public?** Компоненты класса, описанные со спецификатором public, наследуются производным классом как открытые компоненты, которые могут быть использованы в производном классе и за его пределами.
3. **Каким образом наследуются компоненты класса, описанные со спецификатором private?** Компоненты класса, описанные со спецификатором private, наследуются производным классом как закрытые компоненты, к которым производный класс не может обратиться напрямую.
4. **Каким образом наследуются компоненты класса, описанные со спецификатором protected?** Компоненты класса, описанные со спецификатором protected, наследуются производным классом как защищенные компоненты, которые могут быть использованы в производном классе, но не могут быть использованы за пределами производного класса.
5. **Каким образом описывается производный класс?** Для создания производного класса используется ключевое слово "class" или "struct", за которым следует имя производного класса, а после двоеточия - имя базового класса. Например: class Derived : public Base {...};
6. **Наследуются ли конструкторы?** Конструкторы также наследуются от базового класса, но могут быть переопределены в производном классе.
7. **Наследуются ли деструкторы?** Деструкторы также наследуются от базового класса, но могут быть переопределены в производном классе.
8. **В каком порядке конструируются объекты производных классов?** Объекты производных классов конструируются в следующем порядке: сначала конструируется базовый класс, затем производный класс.
9. **В каком порядке уничтожаются объекты производных классов?** Объекты производных классов уничтожаются в обратном порядке: сначала уничтожается производный класс, затем базовый класс.
10. **Что представляют собой виртуальные функции и механизм позднего связывания?** Виртуальные функции - это функции, которые могут быть переопределены в производном классе и вызваны из объекта производного класса, даже если указатель на объект имеет тип базового класса. Механизм позднего связывания позволяет выбрать правильную реализацию виртуальной функции во время выполнения программы.
11. **Могут ли быть виртуальными конструкторы? Деструкторы?** Конструкторы могут быть виртуальными, но деструкторы должны быть виртуальными, если класс имеет хотя бы одну виртуальную функцию.
12. **Наследуется ли спецификатор virtual?** Спецификатор virtual не наследуется.
13. **Какое отношение устанавливает между классами открытое наследование?** Открытое наследование - это тип наследования, при котором наследуемый класс (производный класс) имеет доступ к публичным компонентам наследуемого класса (базового класса), таким как переменные, функции и т.д.
14. **Какое отношение устанавливает между классами закрытое наследование?** Закрытое наследование - это тип наследования, при котором производный класс наследует все компоненты базового класса, но эти компоненты не являются доступными для использования вне производного класса.
15. **В чем заключается принцип подстановки?** Принцип подстановки - это принцип объектно-ориентированного программирования, который гласит, что всякий раз, когда базовый класс может быть использован, производный класс также может быть использован без каких-либо изменений интерфейса.
16. **Имеется иерархия классов: class Student { int age; public: string name; ... }; class Employee : public Student { protected: string post; ... }; class Teacher : public Employee { protected: int stage; ... }; Teacher x; Какие компонентные данные будет иметь объект х?** Объект x будет иметь компонентные данные, определенные в классах Student, Employee и Teacher, включая name, post и stage. age – не будет унаследовано, т.к. это поле в private секции.
17. **Для классов Student, Employee и Teacher написать конструкторы без параметров.**class Student {

public:

Student()   
{

age = 0;

name = "";

}

. . .

class Employee : public Student {  
public:  
Employee()   
{  
 post = "";  
}  
. . .

class Teacher : public Employee {  
public:  
Teacher()  
 {  
 stage = 0;  
}  
. . .

1. **Для классов Student, Employee и Teacher написать конструкторы с параметрами.**class Student {  
   public:  
   Student(int a, string n)   
   {  
    age = a;  
    name = n;  
   }

. . .

class Employee : public Student {  
public:  
Employee(int a, string n, string p) : Student(a, n)   
{  
 post = p;  
}

. . .

class Teacher : public Employee {  
public:  
Teacher(int a, string n, string p, int s) : Employee(a, n, p)   
{  
 stage = s;  
}

. . .

1. **Для классов Student, Employee и Teacher написать конструкторы копирования.**class Student {  
   public:  
   Student(const Student& other)  
    {  
    age = other.age;  
    name = other.name;  
    }

. . .

class Employee : public Student {  
public:

Employee(const Employee& other) : Student(other)   
{

post = other.post;  
}

. . .  
class Teacher : public Employee {  
public:  
Teacher(const Teacher& other) : Employee(other)   
{  
 stage = other.stage;  
}

. . .

1. **Для классов Student, Employee и Teacher определить операцию присваивания.**class Student {   
   Student& operator=(const Student& other)   
   {  
    if (this != &other)   
    {  
    age = other.age;  
    name = other.name;  
    }  
    return \*this;  
   }

. . .

class Employee : public Student {   
Employee & operator=(const Student& other)   
{  
 if (this != &other)   
 {  
 Student::operator=(other);  
 post = other.post;  
 }  
 return \*this;  
}

. . .

class Teacher : public Employee {   
Teacher & operator=(const Student& other)   
{  
 if (this != &other)   
 {  
 Employee::operator=(other);  
 stage = other.stage;  
 }  
 return \*this;  
}

. . .

**Диаграмма класса**

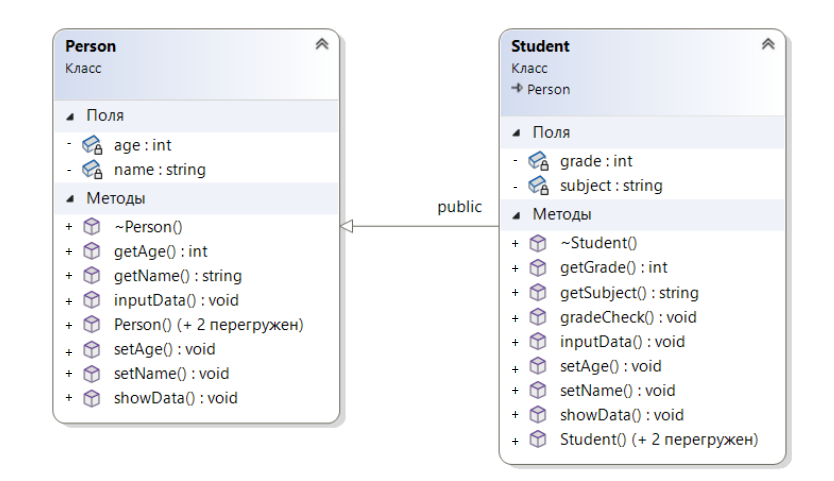


Рисунок 1 – класс Student и Person

**Описание классов**

Class Person

#pragma once

#include <iostream>

#include <string>

#include <iomanip>

using namespace std;

class Person

{

friend class Student;

private:

string name;

int age;

public:

Person();

Person(string name, int age);

Person(const Person& object);

~Person();

void setName(string name) { this->name = name; }

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

string getName() { return name; }

int getAge() { return age; }

virtual void showData();

virtual void inputData();

};

Class Student

#pragma once

#include "Person.h";

class Student : public Person

{

private:

string subject;

int grade;

public:

Student();

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

Student(const Student& object);

~Student();

void setName(string subject) { this->subject = subject; }

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

string getSubject() { return subject; }

int getGrade() { return grade; }

void gradeCheck();

void showData() override;

void inputData() override;

};

**Определение методов**

Class Person

#pragma once

#include "Person.h"

Person::Person()

{

//cout << "Вызвался констуктор по умолчанию для объекта: " << this << endl;

name = "null";

age = 0;

}

Person::Person(string name, int age)

{

//cout << "Вызвался констуктор с параметрами для объекта: " << this << endl;

this->name = name;

this->age = age;

}

Person::Person(const Person& object)

{

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

//cout << "Копирование объекта: " << &object << endl;

this->age = object.age;

this->name = object.name;

}

Person::~Person()

{

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

}

void Person::showData()

{

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

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

}

void Person::inputData()

{

cout << "Введите имя человека: "; cin >> name;

cout << "Введите возраст человека: "; cin >> age;

}

Class Student

#pragma once

#include "Student.h"

Student::Student() : Person()

{

subject = "null";

grade = 0;

}

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

{

this->subject = subject;

this->grade = grade;

}

Student::Student(const Student& object) : Person(object)

{

this->subject = object.subject;

this->grade = object.grade;

}

Student::~Student()

{

}

void Student::gradeCheck()

{

if (grade > 2)

cout << "Оценка удовлетворительная!" << endl;

else

cout << "Неудовлетворительная оценка!" << endl;

}

void Student::showData()

{

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

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

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

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

}

void Student::inputData()

{

cout << "Введите имя студента: "; cin >> name;

cout << "Введите возраст студента: "; cin >> age;

cout << "Введите предмет, по которому будет выставлена оценка: "; cin >> subject;

cout << "Оценка по данному предмету: "; cin >> grade;

}

**Результаты работы программы**

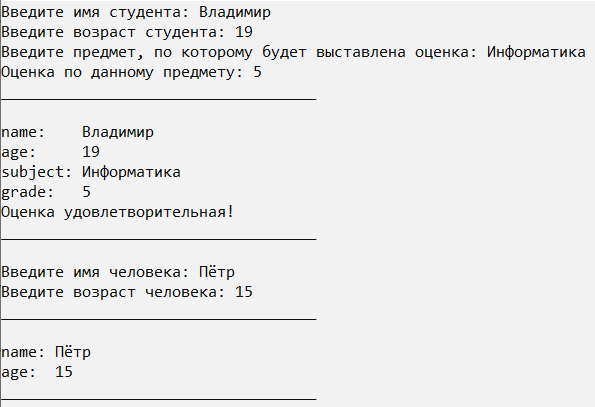


Рисунок 2 – результат работы программы