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 №5

# « Наследование. Виртуальные функции. Полиморфизм »

# 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 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.

**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. **Какой метод называется чисто виртуальным? Чем он отличается от виртуального метода?** Чисто виртуальный метод - это метод, который объявлен в базовом классе, но не имеет реализации в этом классе. Он помечается ключевым словом "pure virtual" или "= 0". Виртуальный метод имеет реализацию в базовом классе, но может быть переопределен в производных классах.
2. **Какой класс называется абстрактным?** Абстрактный класс - это класс, который содержит чисто виртуальные методы и не может быть создан в экземпляре. Он может использоваться только в качестве базового класса для производных классов.
3. **Для чего предназначены абстрактные классы?** Абстрактные классы предназначены для определения интерфейса, который должны реализовывать классы-потомки. Они обеспечивают абстракцию и упрощают процесс проектирования программных систем, так как позволяют определить общие свойства и методы для группы классов.
4. **Что такое полиморфные функции?** Полиморфные функции - это функции, которые могут принимать объекты разных типов и вести с ними согласованно. Это достигается благодаря использованию механизма полиморфизма.
5. **Чем полиморфизм отличается от принципа подстановки?** Полиморфизм - это возможность использования объектов разных типов через общий интерфейс, который позволяет вызывать один и тот же метод с разными объектами. Принцип подстановки - это принцип, который утверждает, что объекты производных классов должны быть совместимы с объектами базового класса. Это позволяет использовать объекты производных классов везде, где используются объекты базового класса.
6. **Привести примеры иерархий с использованием абстрактных классов.** Примеры иерархий с использованием абстрактных классов могут включать иерархии животных, где есть абстрактный класс "Животное" и производные классы "Кошка", "Собака", "Лев" и т.д. В каждом из этих производных классов должны быть реализованы методы, определенные в абстрактном классе "Животное".
7. **Привести примеры полиморфных функций.** Примеры полиморфных функций могут включать функцию вывода информации об объектах разных классов. Например, функция "showInfo" может принимать объекты классов "Кошка", "Собака", "Лев" и т.д., и выводить информацию о каждом объекте.
8. **В каких случаях используется механизм позднего связывания?** Механизм позднего связывания используется в случаях, когда метод должен быть вызван на основе типа объекта во время выполнения программы, а не во время компиляции. Это позволяет использовать полиморфизм и обеспечивает возможность вызова правильной версии метода, определенной в производном классе, в зависимости от типа объекта, на котором метод вызывается. Механизм позднего связывания достигается с помощью виртуальных функций, которые могут быть переопределены в производных классах, и указателей или ссылок на базовый класс, которые могут ссылаться на объекты производных классов. Использование механизма позднего связывания позволяет упростить проектирование и реализацию сложных иерархий классов, обеспечить гибкость и расширяемость кода, а также повысить его переносимость между различными операционными системами и компиляторами.

**Диаграмма класса**

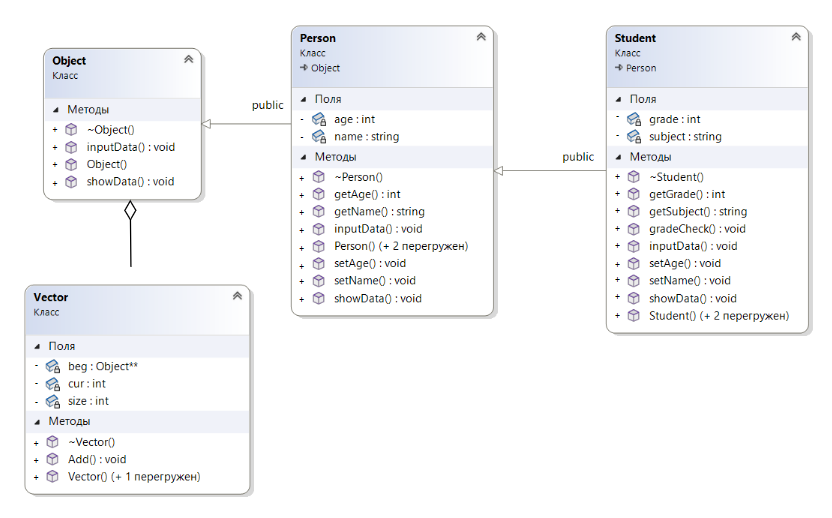


Рисунок 1 – классы Object, Person, Student и Vector

**Описание классов**

Class Object

#pragma once

#include <iostream>

#include <string>

#include <iomanip>

using namespace std;

class Object

{

public:

Object() {};

virtual void showData() = 0;

virtual void inputData() = 0;

virtual ~Object() = 0;

};

Class Person

#pragma once

#include "Objects.h"

class Person : public Object

{

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

friend class Student;

private:

string name;

int age;

public:

Person();

Person(string name, int age);

Person(const Person& object);

~Person() override;

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() override;

virtual void inputData() override;

};

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() override;

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 Objects

#include "Objects.h"

Object::~Object()

{

}

Class Person

#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;

}

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**

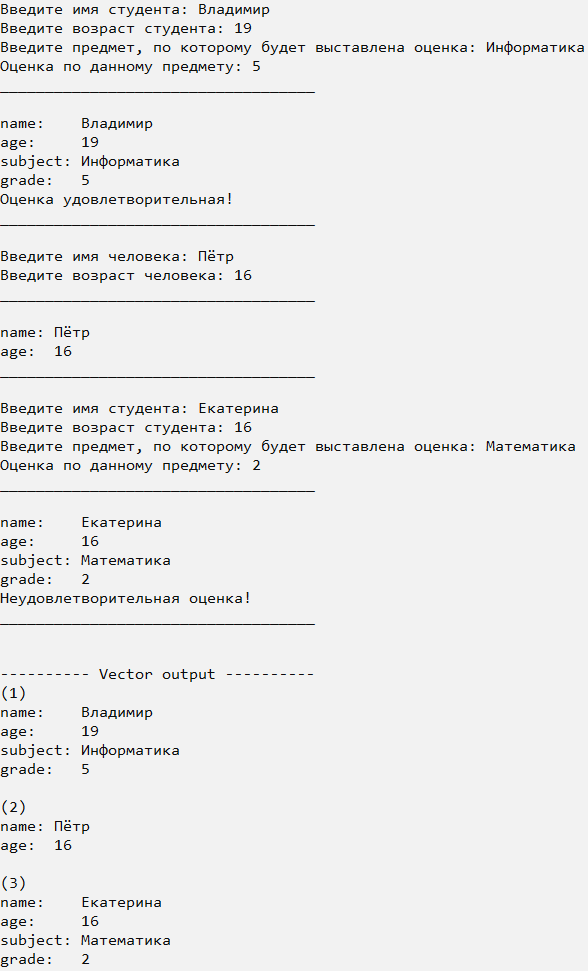


Рисунок 2 – результат работы программы