Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
Пермский национальный исследовательский политехнический университет
Электротехнический факультет
Кафедра информационных технологий и автоматизированных систем

Лабораторная работа №5

" Наследование. Виртуальные функции. Полиморфизм "

Вариант: 12

Выполнил студент ИВТ-24-2б:
Шишкин Максим Григорьевич

_____
(дата, подпись)

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

_____
(дата, подпись)

Пермь 2025

Содержание

1 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: Базовый класс:

ЧЕЛОВЕК (PERSON)

Имя (name) – string

Возраст (age) – int

Определить методы изменения полей. Создать производный класс STUDENT, имеющий поле год обучения. Определить методы изменения и увеличения года обучения.

2 Код на C++

Person.h | Person.cpp | Student.h | Student.cpp | PersonVector.h | PersonVector.cpp | main.cpp

5лабапеределка — (Глобальная область)

```cpp
#include <clocale>
#include <windows.h>
#include "Student.h"
#include "PersonVector.h"

int main() {
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    setlocale(LC_ALL, "Russian");

    PersonVector pv;

    pv.add(new Student("Иван Иванов", 20, 2));
    pv.add(new Student("Петр Петров", 21, 3));
    pv.add(new Student("Анна Сидорова", 19, 1));

    cout << "Содержимое вектора:" << endl;
    cout << pv;

    pv.demonstratePolymorphism();

    Student* s = new Student("Мария Кузнецова", 22, 4);
    pv.add(s);

    cout << "\nПосле изменения данных:" << endl;
    s->setName("Мария Смирнова");
    s->setAge(23);
    s->incrementStudyYear();

    cout << pv;

    return 0;
}
```

(int)19
Поиск в Интернете

Person.h | Person.cpp | Student.h | Student.cpp | PersonVector.h | PersonVector.cpp | main.cpp

5лабапеределка — (Глобальная область)

```cpp
#include "PersonVector.h"
#include <iostream>

PersonVector::~PersonVector() {
    for (Person* p : items) {
        delete p;
    }
}

void PersonVector::add(Person* p) {
    items.push_back(p);
}

ostream& operator<<(ostream& os, const PersonVector& pv) {
    for (const Person* item : pv.items) {
        os << *item << endl;
    }
    return os;
}

void PersonVector::demonstratePolymorphism() {
    if (!items.empty()) {
        cout << "Демонстрация полиморфизма:" << endl;
        items[0]->printInfo();
        cout << endl;
    }
}
```

Person.h | Person.cpp | Student.h | Student.cpp | **PersonVector.h** | PersonVector.cpp | main.cpp

5лабапеределка — (Глобальная область)

```cpp
#pragma once
#include <vector>
#include "Person.h"

class PersonVector {
private:
    vector<Person*> items;

public:
    ~PersonVector();

    void add(Person* p);

    friend ostream& operator<<(ostream& os, const PersonVector& pv);

    void demonstratePolymorphism();
};
```

Person.h | Person.cpp | Student.h | **Student.cpp** | PersonVector.h | PersonVector.cpp | main.cpp

5лабапеределка — (Глобальная область)

```cpp
#include "Student.h"

Student::Student(const string& n, int a, int year) : Person(n, a), studyYear(year) {}

void Student::printInfo() const {
    cout << "Студент: " << name << ", Возраст: " << age << ", Год обучения: " << studyYear;
}

void Student::setStudyYear(int year) {
    studyYear = year;
}

void Student::incrementStudyYear() {
    studyYear++;
}
```

Person.h | Person.cpp | **Student.h** | Student.cpp | PersonVector.h | PersonVector.cpp | main.cpp

5лабапеределка — (Глобальная область)

```cpp
#pragma once
#include "Person.h"

class Student : public Person {
private:
    int studyYear;

public:
    Student(const string& n, int a, int year);

    void printInfo() const override;

    void setStudyYear(int year);
    void incrementStudyYear();
};
```

Person.h | **Person.cpp** | Student.h | Student.cpp | PersonVector.h | PersonVector.cpp | main.cpp

5лабапеределка — (Глобальная область)

```cpp
#include "Person.h"

Person::Person(const string& n, int a) : name(n), age(a) {}

Person::~Person() {}

void Person::setName(const string& n) {
    name = n;
}

void Person::setAge(int a) {
    age = a;
}

ostream& operator<<(ostream& os, const Person& p) {
    p.printInfo();
    return os;
}
```

Person.h | Person.cpp | Student.h | Student.cpp | PersonVector.h | PersonVector.cpp | main.cpp

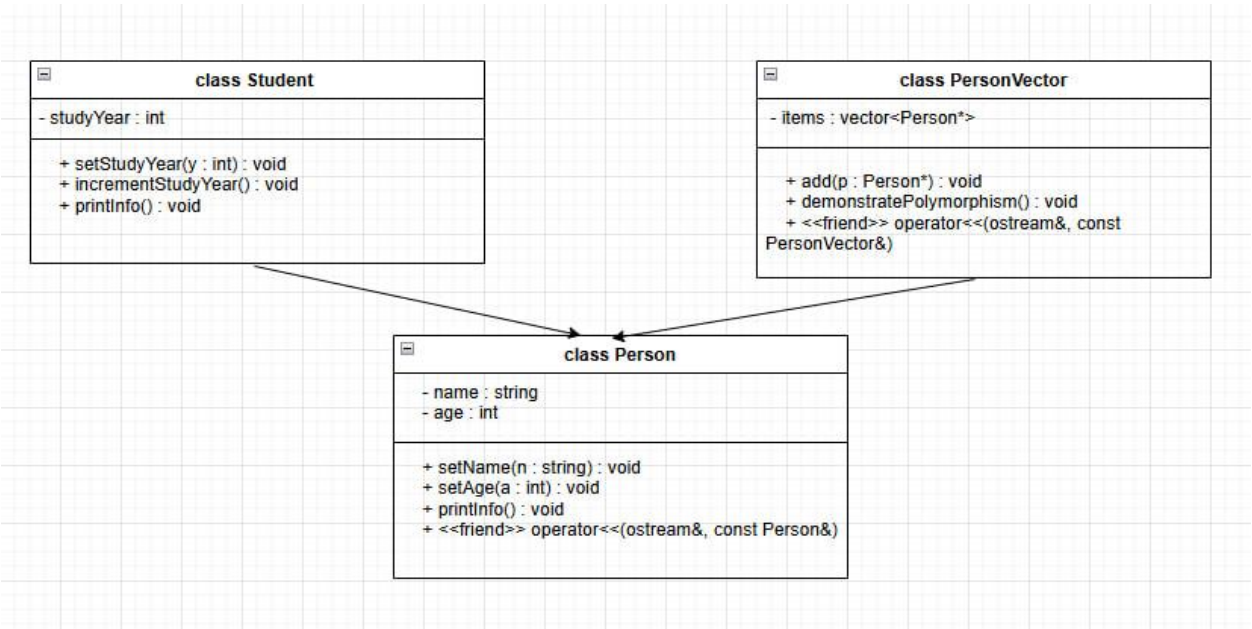
5лабапеределка — (Глобальная область)

```cpp
#pragma once
#include <iostream>
#include <string>

using namespace std;

class Person {
protected:
    string name;
    int age;

public:
    Person(const string& n, int a);
    virtual ~Person();

    virtual void printInfo() const = 0;

    void setName(const string& n);
    void setAge(int a);

    friend ostream& operator<<(ostream& os, const Person& p);
};
```

3 Результаты работы

```
Содержимое вектора:
Студент: Иван Иванов, Возраст: 20, Год обучения: 2
Студент: Петр Петров, Возраст: 21, Год обучения: 3
Студент: Анна Сидорова, Возраст: 19, Год обучения: 1
Демонстрация полиморфизма:
Студент: Иван Иванов, Возраст: 20, Год обучения: 2

После изменения данных:
Студент: Иван Иванов, Возраст: 20, Год обучения: 2
Студент: Петр Петров, Возраст: 21, Год обучения: 3
Студент: Анна Сидорова, Возраст: 19, Год обучения: 1
Студент: Мария Смирнова, Возраст: 23, Год обучения: 5
```

4 UML-диаграмма классов



5 Ответы на контрольные вопросы

1. Чисто виртуальным называется метод, объявленный с использованием синтаксиса = 0. В отличие от обычного виртуального метода, он **не имеет реализации в базовом классе** и **обязательно должен быть переопределён в производном классе**, чтобы можно было создать объект этого класса.

2. Абстрактным называется класс, в котором есть хотя бы один чисто виртуальный метод. Такие классы нельзя инстанцировать (создавать объекты), они используются только как основа для наследования.

3. Абстрактные классы предназначены для **задания интерфейса** или **общего поведения**, которое должно быть реализовано в производных классах. Они позволяют задать структуру и требования к функциональности наследников.

4. Полиморфные функции — это виртуальные функции, которые могут **иметь разную реализацию** в разных производных классах, но **вызываются через указатель или ссылку на базовый класс**. Реализация, которая будет вызвана, определяется во время выполнения (run-time).

5. Полиморфизм — это возможность вызывать методы производных классов через интерфейс базового класса.
Принцип подстановки Лисков заключается в том, что объект производного класса может использоваться везде, где ожидается объект базового класса, **без изменения корректности программы**. Полиморфизм — это один из механизмов, обеспечивающий этот принцип.

6. class Shape {

public:

```
virtual void draw() const = 0;
```

};

class Circle : public Shape {

public:

```
void draw() const override {
    cout << "Drawing a circle" << endl;
}
```

};

7. void showInfo(const Person* p) {

p->printInfo(); // вызов будет зависеть от типа объекта (Student, Teacher и т.д.)

}

8. Механизм позднего связывания используется, когда метод объявлен как virtual и вызывается через указатель или ссылку на базовый класс. Он позволяет вызывать переопределённую версию метода из производного класса в момент выполнения (run-time), а не компиляции.

6 Ссылка на github

ссылка на github - https://github.com/MAKSPOWERO/mas1