Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа №5**

**«Наследование. Виртуальные функции. Полиморфизм»**

**14 вариант**

Выполнил:

Студент группы РИС-23-1б

Шароглазов Егор Алексеевич

Проверила:

Доцент кафедры ИТАС

О.А. Полякова

1. г

**Постановка задачи:**

1. Определить абстрактный класс.
2. Определить иерархию классов, в основе которой будет находиться абстрактный класс.
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.

**Анализ задачи:**

**1. Определение абстрактного класса PERSON:**

* Определить атрибуты: name (строка) и age (целое число).
* Определить виртуальные методы для изменения атрибутов: setName и setAge.
* Определить чистый виртуальный метод printInfo для вывода информации.

**2. Определение производного класса TEACHER:**

* Наследовать от PERSON.
* Определить дополнительные атрибуты: subject (строка) и hours (целое число).
* Переопределить методы setName и setAge для обновления производных атрибутов.
* Определить методы для изменения поля предмета и часов: setSubject, setHours, addHours и reduceHours.
* Переопределить метод printInfo для вывода дополнительной информации о предмете и часах.

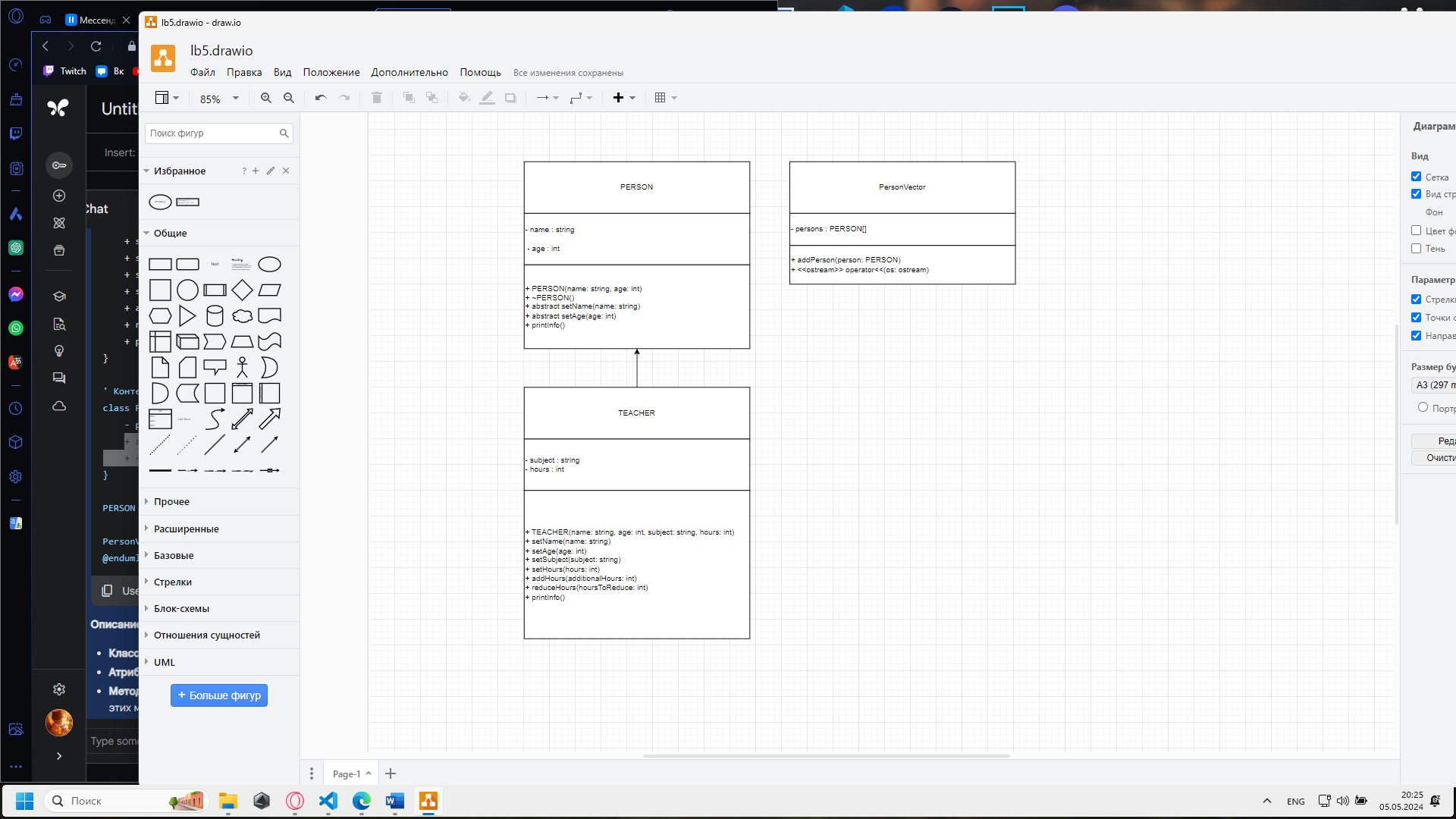
**3. Определение класса PersonVector:**

* Определить динамический массив persons для хранения указателей на объекты PERSON.
* Определить метод addPerson для добавления указателя на объект в массив.
* Перегрузить оператор вывода << для итерации по массиву и вызова printInfo для каждого объекта.

**4. Демонстрация перегруженного оператора и полиморфизма в главной функции:**

* Создать объекты TEACHER.
* Добавить объекты в PersonVector.
* Вызвать методы для изменения полей.
* Вывести информацию о всех объектах в векторе с помощью перегруженного оператора вывода.

**UML - диаграмма**



**Код**

#include <iostream>

#include <string>

#include <vector>

// Абстрактный базовый класс PERSON

class PERSON {

protected:

    std::string name;

    int age;

public:

    PERSON(const std::string& name, int age) : name(name), age(age) {}

    virtual ~PERSON() {}

    // Виртуальные методы для изменения полей (чисто виртуальные - абстрактные)

    virtual void setName(const std::string& name) = 0;

    virtual void setAge(int age) = 0;

    // Метод для вывода информации

    virtual void printInfo() const {

        std::cout << "Имя: " << name << ", Возраст: " << age;

    }

};

// Производный класс TEACHER

class TEACHER : public PERSON {

private:

    std::string subject;

    int hours;

public:

    TEACHER(const std::string& name, int age, const std::string& subject, int hours)

        : PERSON(name, age), subject(subject), hours(hours) {}

    // Переопределение виртуальных методов

    void setName(const std::string& name) override { this->name = name; }

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

    // Методы для изменения полей предмета и часов

    void setSubject(const std::string& subject) { this->subject = subject; }

    void setHours(int hours) { this->hours = hours; }

    void addHours(int additionalHours) { hours += additionalHours; }

    void reduceHours(int hoursToReduce) {

        if (hoursToReduce <= hours) {

            hours -= hoursToReduce;

        } else {

            std::cout << "Попытка уменьшить большее количество часов, чем их указано" << std::endl;

        }

    }

    // Переопределение метода вывода информации

    void printInfo() const override {

        PERSON::printInfo(); // Вызов метода базового класса

        std::cout << ", Предмет: " << subject << ", Часы: " << hours << std::endl;

    }

};

// Класс Vector для хранения указателей на объекты PERSON

class PersonVector {

private:

    std::vector<PERSON\*> persons;

public:

    // Добавление объекта в вектор

    void addPerson(PERSON\* person) {

        persons.push\_back(person);

    }

    // Перегрузка оператора вывода

    friend std::ostream& operator<<(std::ostream& os, const PersonVector& vector) {

        for (const PERSON\* person : vector.persons) {

            person->printInfo(); // Полиморфный вызов printInfo

        }

        return os;

    }

};

int main() {

    // Создание объектов

    TEACHER teacher1("Иван Петрович", 50, "Математика", 20);

    TEACHER teacher2("Мария Ивановна", 45, "Физика", 18);

    // Создание вектора и добавление объектов

    PersonVector personVector;

    personVector.addPerson(&teacher1);

    personVector.addPerson(&teacher2);

    // Вывод информации о всех объектах в векторе

    std::cout << personVector << std::endl;

    // Использование методов

    teacher1.setSubject("Алгебра"); // Изменение предмета для учителя 1

    teacher1.addHours(5); // Увеличение часов для учителя 2

    teacher2.reduceHours(3); // Уменьшение часов для учителя 2

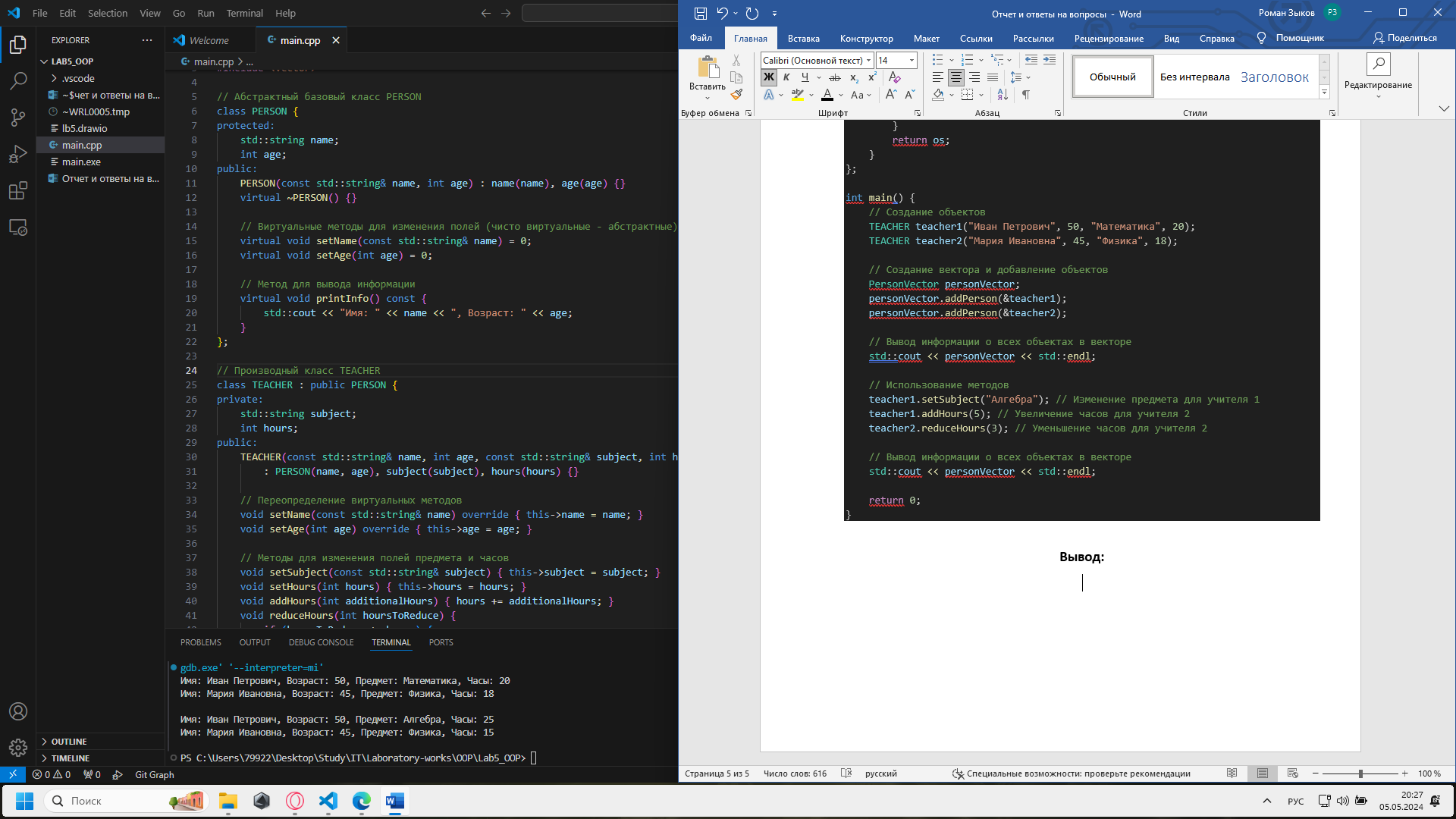
    // Вывод информации о всех объектах в векторе

    std::cout << personVector << std::endl;

    return 0;

}

**Вывод:**



**Вопросы:**

1. **Чисто виртуальный метод** - это метод в абстрактном классе, который объявлен как virtual и не имеет реализации в базовом классе. Он обязывает производные классы реализовать этот метод. Отличие от обычного виртуального метода заключается в том, что чисто виртуальный метод не имеет реализации в базовом классе, он просто определяет интерфейс.
2. **Абстрактный класс** - это класс, содержащий хотя бы один чисто виртуальный метод. Это означает, что нельзя создать объект абстрактного класса напрямую, только его производные классы, реализующие все чисто виртуальные методы.
3. **Абстрактные классы** предназначены для создания общего интерфейса, который должны реализовать конкретные классы-наследники. Они позволяют абстрагироваться от конкретной реализации и работать с классами по общим методам и свойствам.
4. **Полиморфные функции** - это функции, которые могут принимать объекты различных классов в качестве параметров, а также возвращать указатели или ссылки на объекты базового класса. Это позволяет использовать одну и ту же функцию для работы с объектами разных типов.
5. **Полиморфизм** - это способность объекта обрабатывать данные различных типов. Принцип подстановки (Liskov Substitution Principle) гласит, что объекты базового класса могут быть заменены объектами его производных классов без изменения желаемых свойств программы. Таким образом, полиморфизм является концепцией, а принцип подстановки - его реализацией в объектно-ориентированном программировании.
6. Примеры иерархий с использованием абстрактных классов:
   * Shape (Фигура) с чисто виртуальным методом draw() и производными классами Circle (Круг), Rectangle (Прямоугольник), Triangle (Треугольник) и т. д.
   * Animal (Животное) с чисто виртуальным методом makeSound() и производными классами Dog (Собака), Cat (Кошка), Bird (Птица) и т. д.
7. Примеры полиморфных функций:
   * Функция printArea(Shape\* shape) для вывода площади фигуры, где Shape - абстрактный класс.
   * Функция calculatePrice(Vehicle\* vehicle) для расчета цены транспортного средства, где Vehicle - абстрактный класс.
8. Механизм **позднего связывания** используется в случае, когда нам нужно определить, какая версия метода будет вызвана во время выполнения программы. Это особенно важно, когда методы переопределены в производных классах. Используется в случае использования виртуальных функций и указателей на объекты базового класса, которые могут указывать на объекты производных классов.