Пермский национальный исследовательский политехнический университет.

Лабораторная работа № 4 по ООП.

«Простое наследование. Принцип подстановки».

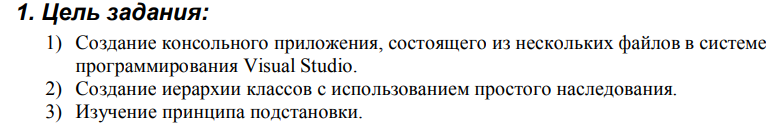
Выполнил: студент группы РИС-23-2б

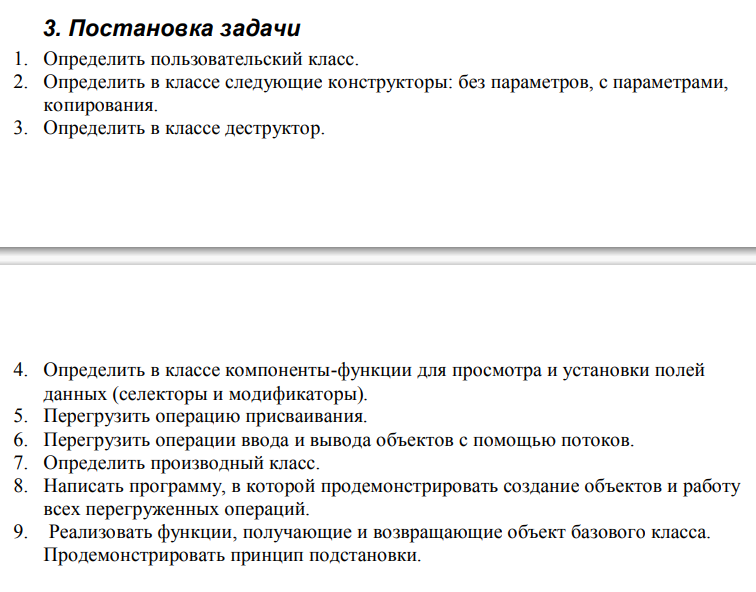
Вековшинин Иван Николаевич

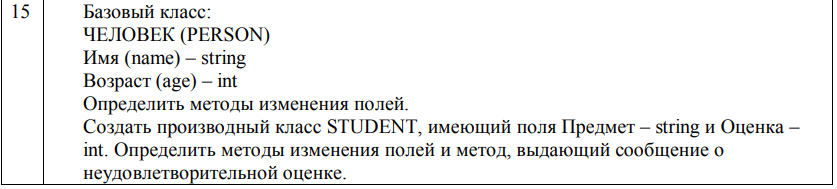
Проверила: доцент кафедры ИТАС

О.А. Полякова.

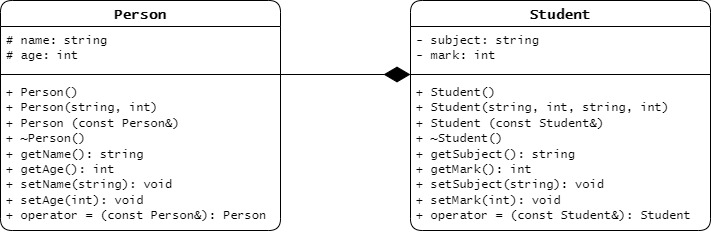
2024 г.

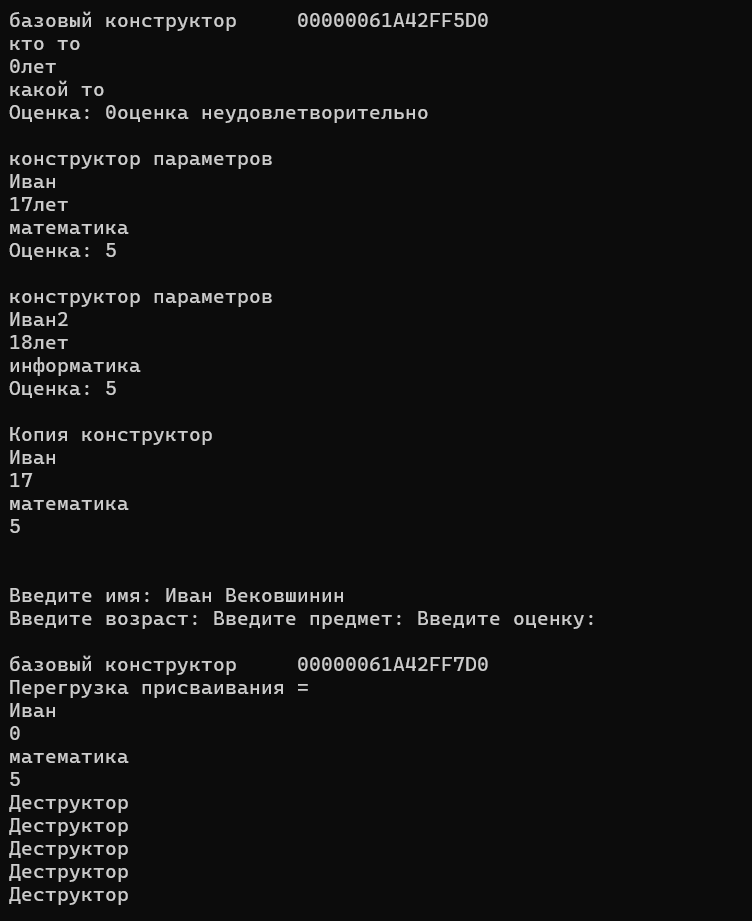
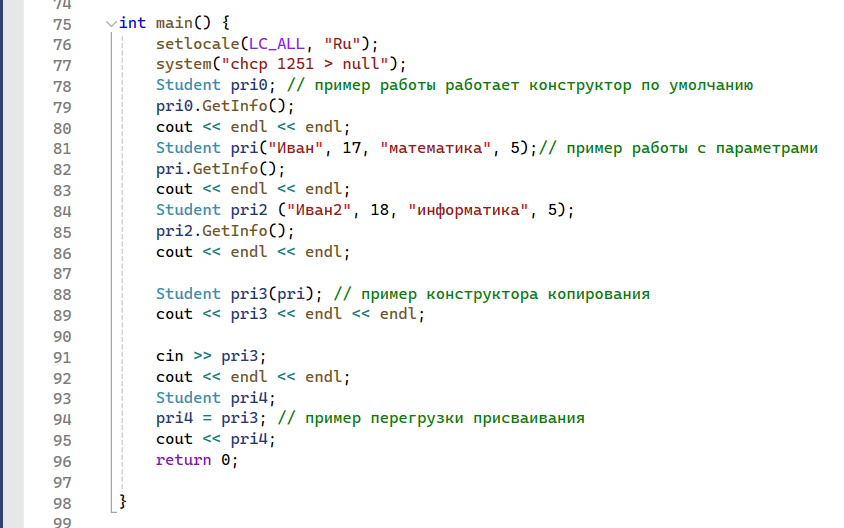
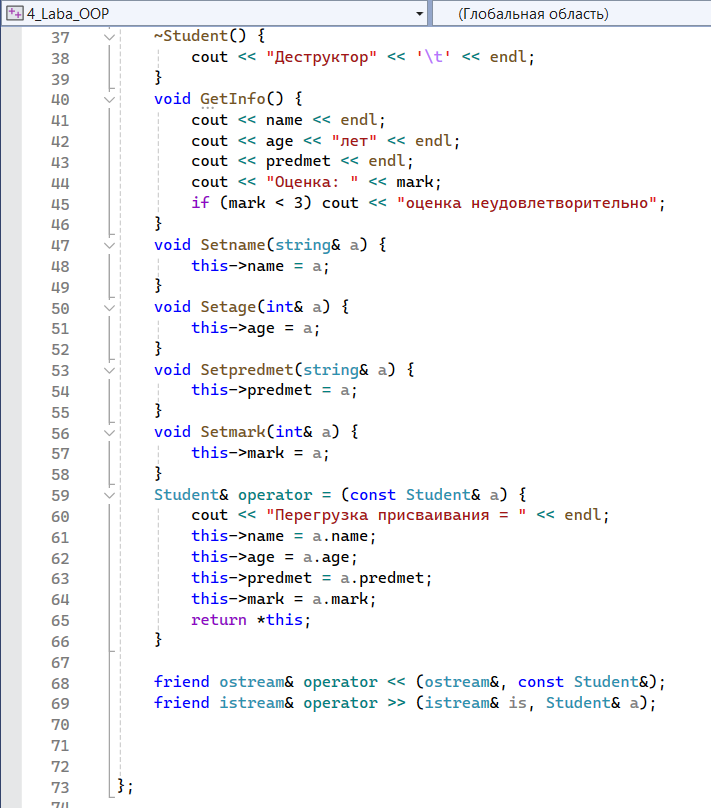
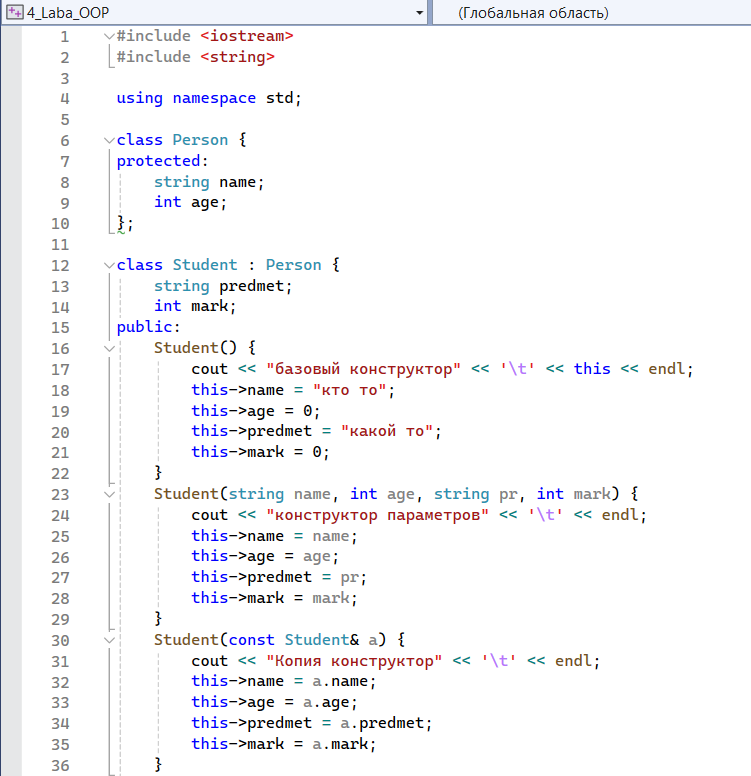




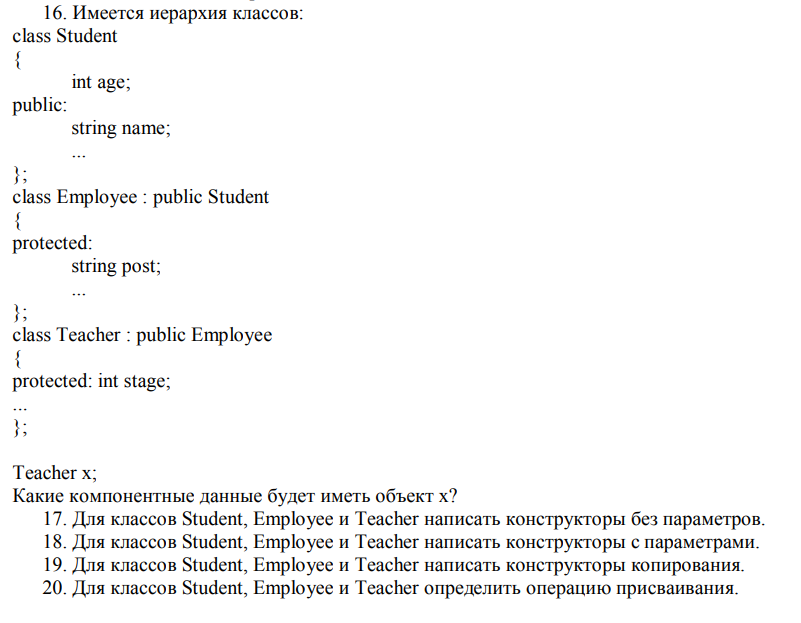
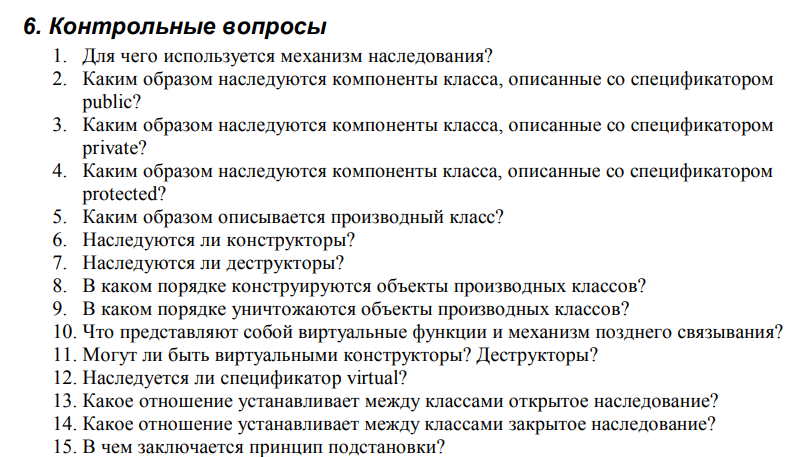


UML Diagram





Ответы на вопросы:



1. Механизм наследования в языке программирования C++ используется для создания иерархии классов, где производные классы могут наследовать свойства и методы базовых классов.

2. Компоненты класса, описанные со спецификатором public, наследуются производным классом таким же способом, как они были объявлены в базовом классе - они остаются public в производном классе.

3. Компоненты класса, описанные со спецификатором private, не наследуются производным классом и становятся недоступными в производном классе.

4. Компоненты класса, описанные со спецификатором protected, наследуются производным классом и становятся доступными в производном классе как protected.

5. Производный класс описывается путем указания базового класса в определении производного класса с использованием символа :.

6. Конструкторы не наследуются, но вызываются при создании объекта производного класса.

7. Деструкторы не наследуются, но вызываются при уничтожении объекта производного класса.

8. Объекты производных классов конструируются в порядке от базового к производному.

9. Объекты производных классов уничтожаются в порядке от производного к базовому.

10. Виртуальные функции позволяют реализовать механизм позднего связывания, когда вызываемый метод определяется во время выполнения программы, а не во время компиляции.

11. Конструкторы могут быть виртуальными (виртуальные конструкторы), но деструкторы должны быть виртуальными, чтобы обеспечить правильное освобождение памяти при удалении объектов через указатель на базовый класс.

12. Спецификатор virtual не наследуется, поэтому если метод был объявлен в базовом классе как виртуальный, он остается виртуальным и в производных классах.

13. Открытое наследование устанавливает "является" отношение между базовым и производным классами, где производный класс является расширением базового.

14. Закрытое наследование устанавливает "реализуется в терминах" отношение между базовым и производным классами, где производный класс реализует интерфейс базового класса.

15. Принцип подстановки (Liskov Substitution Principle) заключается в том, что объекты производного класса могут быть использованы везде, где используются объекты базового класса без изменения поведения программы.

16. Объект x класса Teacher будет иметь следующие компонентные данные:

1. Поля из класса Student:

- int age;

- string name;

2. Поля из класса Employee:

- string post;

3. Поля из класса Teacher:

- int stage;

Таким образом, объект x будет содержать все поля, унаследованные от классов Student и Employee, а также свои собственные поля.

17.

class Student {

public:

Student() {

// Конструктор без параметров для класса Student

}

};

class Employee {

public:

Employee() {

// Конструктор без параметров для класса Employee

}

};

class Teacher {

public:

Teacher() {

// Конструктор без параметров для класса Teacher

}

};

18.

class Student {

public:

Student(string name, int id) {

// Конструктор с параметрами для класса Student

}

};

class Employee {

public:

Employee(string name, int employeeId) {

// Конструктор с параметрами для класса Employee

}

};

class Teacher {

public:

Teacher(string name, string subject) {

// Конструктор с параметрами для класса Teacher

}

};

19.

class Student {

public:

Student(const Student& other) {

// Конструктор копирования для класса Student

}

};

class Employee {

public:

Employee(const Employee& other) {

// Конструктор копирования для класса Employee

}

};

class Teacher {

public:

Teacher(const Teacher& other) {

// Конструктор копирования для класса Teacher

}

};

20.

class Student {

public:

Student& operator=(const Student& other) {

// Операция присваивания для класса Student

return \*this;

}

};

class Employee {

public:

Employee& operator=(const Employee& other) {

// Операция присваивания для класса Employee

return \*this;

}

};

class Teacher {

public:

Teacher& operator=(const Teacher& other) {

// Операция присваивания для класса Teacher

return \*this;

}

};