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

ПНИПУ

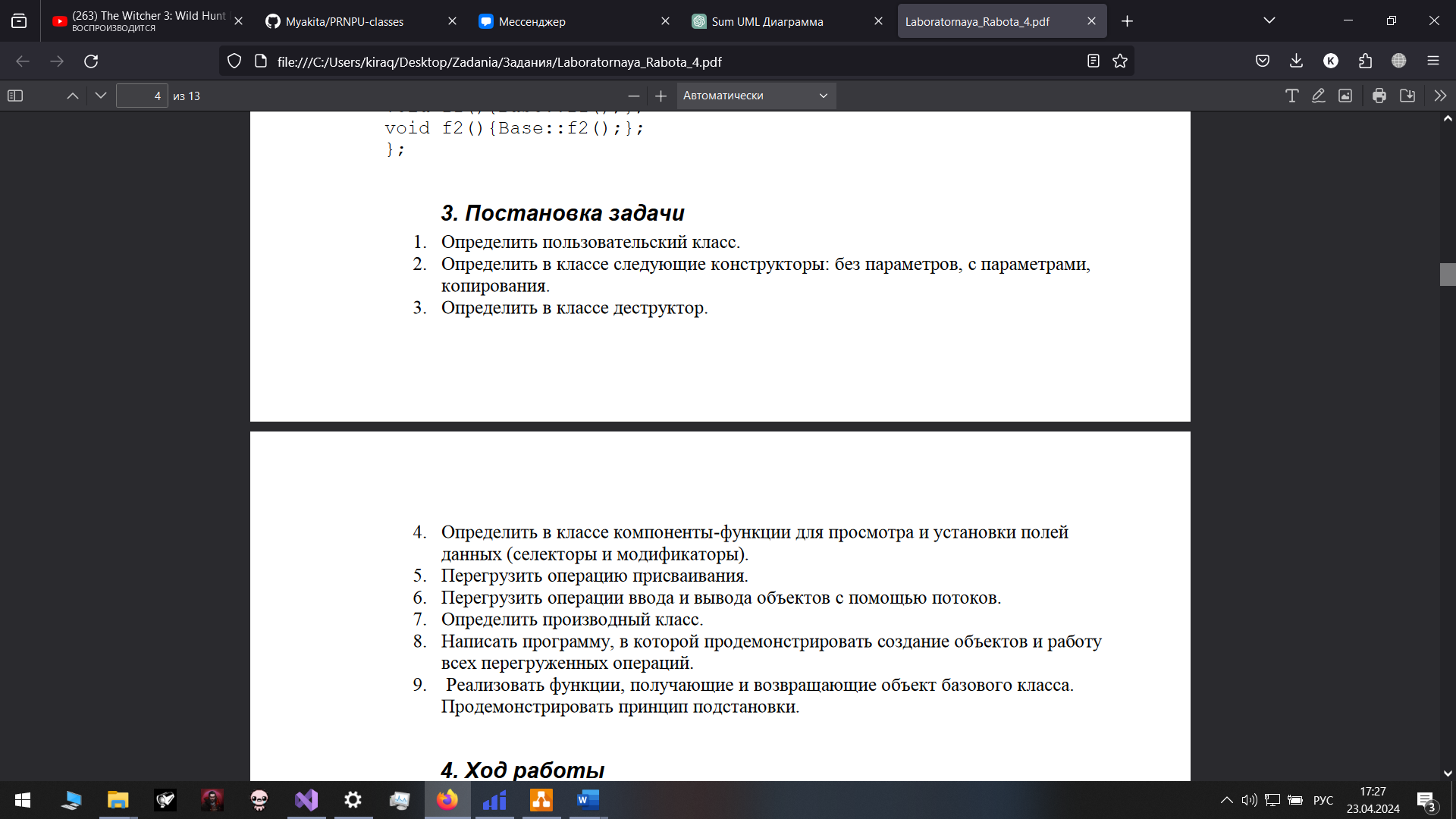
Лабораторная работа  
«Наследование»

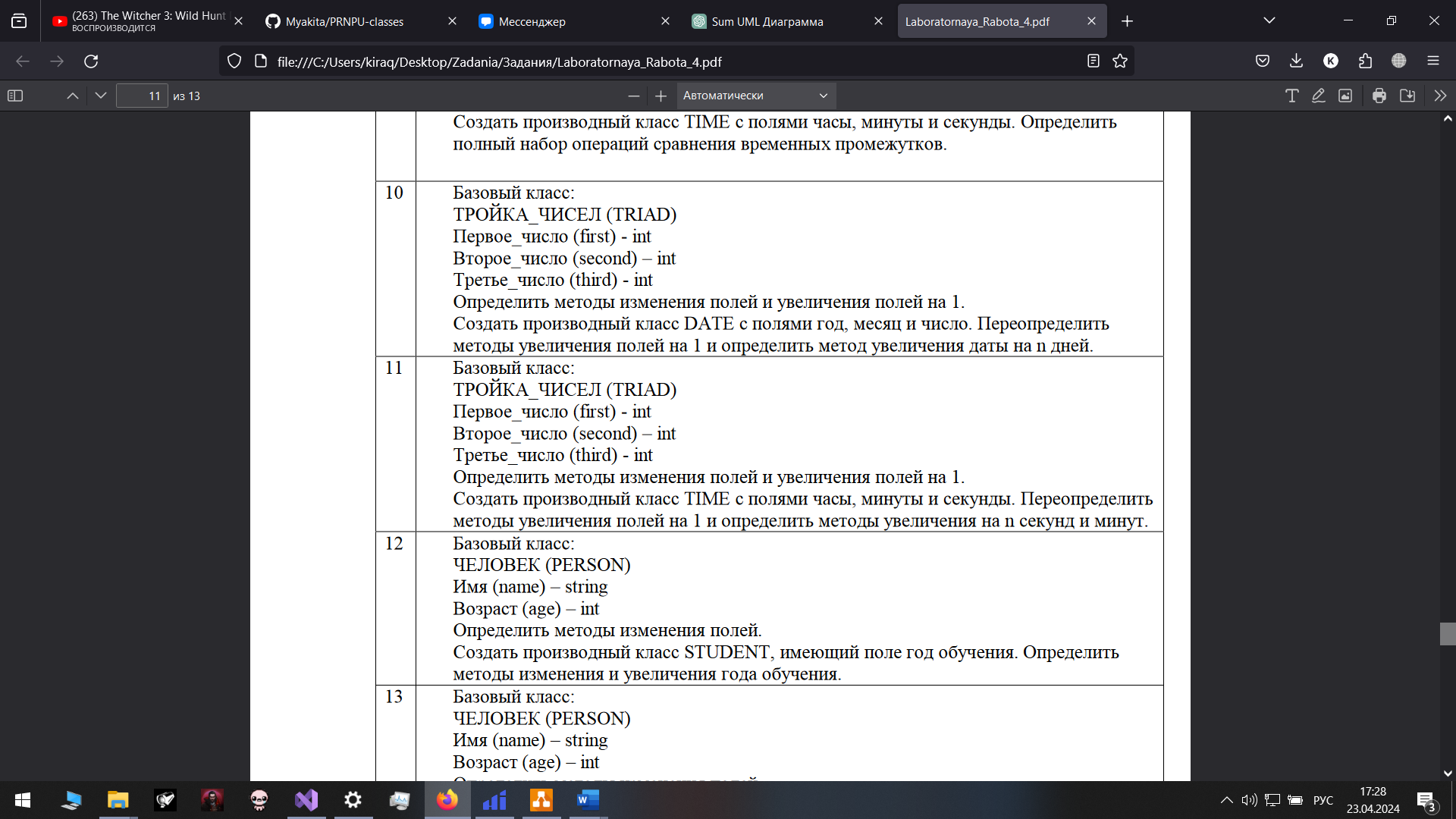
Выполнил:   
студент группы РИС-23-1б   
Мокрушин Никита Дмитриевич

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

Пермь 2024 г.

Постановка задачи:





Анализ задачи:

1. **Конструкторы и деструкторы**:
   * Класс triad имеет конструкторы по умолчанию, с параметрами и копирования. Деструктор класса triad не определен явно, поэтому используется деструктор по умолчанию.
   * Класс Time также имеет конструкторы по умолчанию, с параметрами и копирования. Деструктор класса Time определен явно, но не реализован в данном коде.
2. **Методы и перегрузка операций**:
   * Оба класса содержат методы для получения и установки значений атрибутов, а также перегрузку операторов ввода/вывода.
   * Класс Time расширяет класс triad, добавляя атрибуты для хранения времени (часы, минуты, секунды) и методы для работы с временем (например, plus1 и plusn для увеличения времени на 1 секунду или на произвольное количество секунд).
3. **Функции и операции**:
   * В функции func1 изменяются атрибуты объекта класса triad.
   * Функция func2 создает объект класса Time и возвращает его как объект класса triad.
4. **Основная функция main**:
   * Создаются объекты классов triad и Time.
   * Пользователь вводит параметры для объектов, выполняются операции присваивания и вывод результатов.

Код:

#include <iostream>

using namespace std;

class triad

{

public:

triad() : first(0), second(0), third(0) {}

triad(int f, int s, int t) : first(f), second(s), third(t) {}

triad(const triad& t) : first(t.first), second(t.second), third(t.third) {}

~triad(); // ВОТ ЗДЕСЬ

int getfirst() const { return first; }

int getsecond() const { return second; }

int getthird() const { return third; }

void setfirst(int f) { first = f; }

void setsecond(int s) { second = s; }

void setthird(int t) { third = t; }

triad& operator=(const triad& t)

{

if (this != &t)

{

first = t.first;

second = t.second;

third = t.third;

}

return \*this;

}

friend ostream& operator<<(ostream& os, const triad& t)

{

os << "\nfirst:" << t.first;

os << "\nsecond:" << t.second;

os << "\nthird:" << t.third;

return os;

}

friend istream& operator>>(istream& is, triad& t)

{

is >> t.first >> t.second >> t.third;

return is;

}

private:

int first;

int second;

int third;

};

// ВОТ ЗДЕСЬ

triad::~triad() {}

class Time : public triad

{

public:

Time() {}

Time(int h, int m, int s) : triad(h, m, s)

{

hours = h;

minutes = m;

seconds = s;

}

Time(const Time& tm)

{

hours = tm.hours;

minutes = tm.minutes;

seconds = tm.seconds;

}

int gethours() const { return hours; }

int getminutes() const { return minutes; }

int getseconds() const { return seconds; }

void sethours(int f) { hours = f; }

void setsecond(int s) { minutes = s; }

void setthird(int t) { seconds = t; }

~Time(); // ВОТ ЗДЕСЬ

triad& operator=(const Time& tm)

{

if (this != &tm)

{

hours = tm.hours;

minutes = tm.minutes;

seconds = tm.seconds;

}

return \*this;

}

friend ostream& operator<<(ostream& os, const Time& tm)

{

os << "\nHours:" << tm.hours;

os << "\nMinutes:" << tm.minutes;

os << "\nSeconds:" << tm.seconds;

return os;

}

friend istream& operator>>(istream& is, Time& tm)

{

is >> tm.hours >> tm.minutes >> tm.seconds;

return is;

}

void plus1(Time& t)

{

t.seconds = t.seconds + 1;

if (t.seconds == 60)

{

t.minutes = t.minutes + 1;

if (t.minutes == 60)

{

t.hours = t.hours + 1;

}

}

cout << t;

}

void plusn(Time& t, int n)

{

t.seconds = t.seconds + n;

if (t.seconds == 60)

{

t.minutes = t.minutes + n;

if (t.minutes == 60)

{

t.hours = t.hours + n;

}

}

cout << t;

}

private:

int hours;

int minutes;

int seconds;

};

// ВОТ ЗДЕСЬ

Time::~Time() {}

void func1(triad& t)

{

t.setfirst(1);

t.setsecond(4);

t.setthird(88);

cout << "Вывод функции func1:" << t;

}

triad func2()

{

Time tm(1, 4, 88);

return tm;

}

int main()

{

setlocale(0, "");

system("chcp 1251>nul");

triad t;

cout << "Введите параметры для триады: ";

cin >> t;

cout << "Класс триады: " << t << endl;

triad b(1, 4, 88);

cout << "Заданные параметры триады, но другого объекта:" << b << endl;

t = b;

cout << "Присваивание объекта b объектом t" << t << endl;

Time tm;

cout << "Введите время: ";

cin >> tm;

cout << "Время:" << tm << endl;

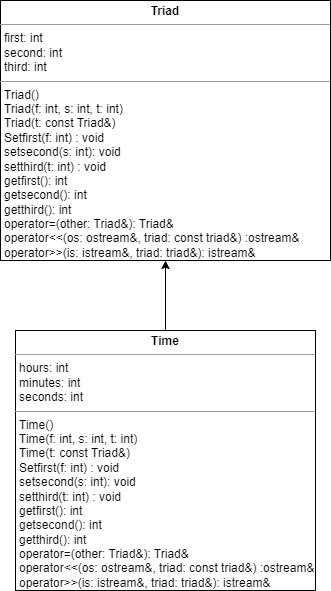
func1(t);

t = func2();

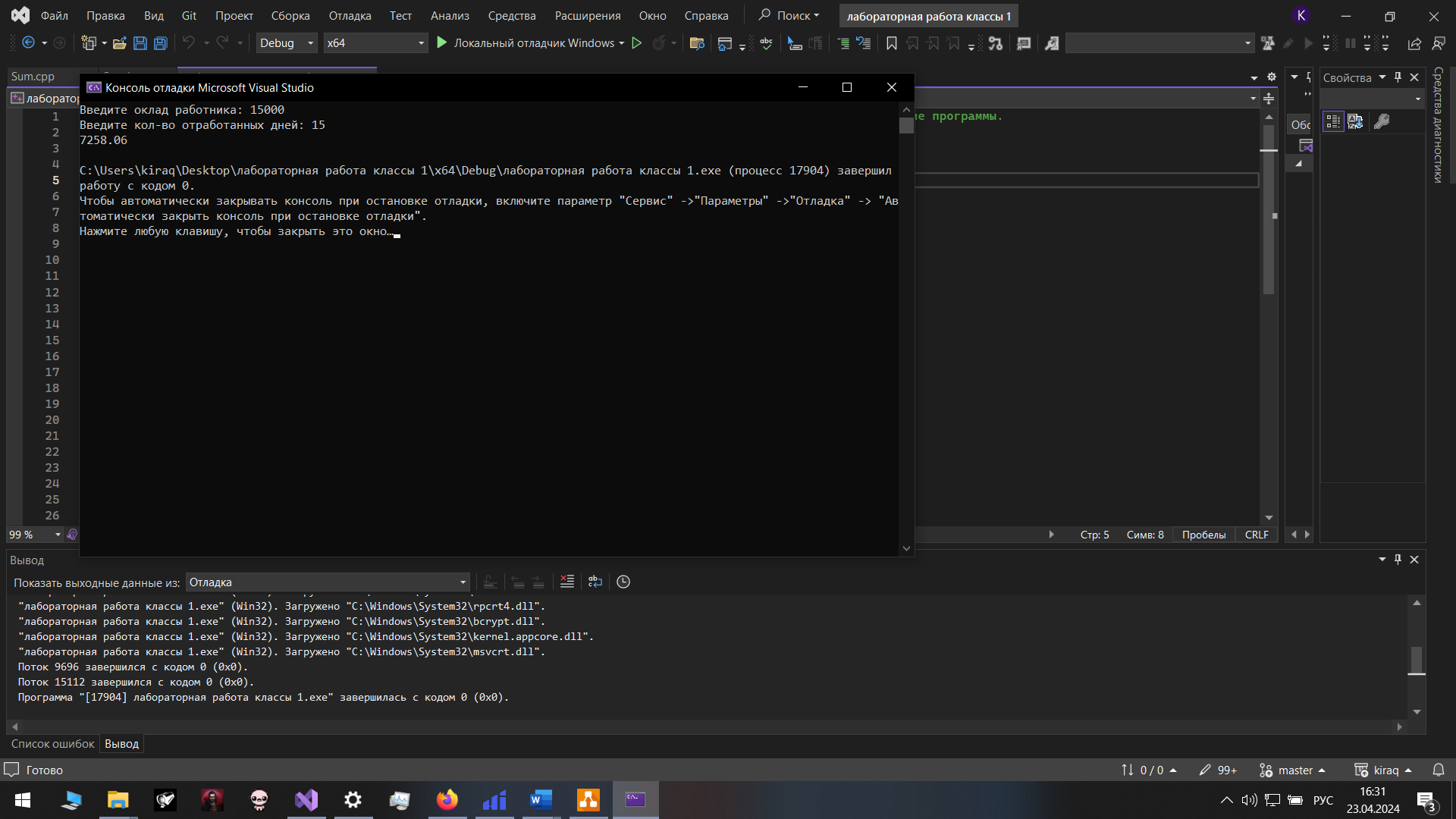
cout << t << endl;

}

UML Диаграммы:



Вывод:



программа работает корректно и выдаёт требуемый результат.  
Ответы на вопросы:

 **Использование механизма наследования**:

* Механизм наследования в языке программирования C++ позволяет создавать новые классы на основе уже существующих. Производный класс получает доступ к членам базового класса, что способствует повторному использованию кода и упрощает разработку и поддержку программ.

 **Наследование компонентов класса с public-спецификатором**:

* Компоненты класса, описанные со спецификатором public, наследуются в производном классе с тем же самым уровнем доступа.

 **Наследование компонентов класса с private-спецификатором**:

* Компоненты класса, описанные со спецификатором private, наследуются в производном классе как private, то есть они не доступны напрямую в производном классе.

 **Наследование компонентов класса с protected-спецификатором**:

* Компоненты класса, описанные со спецификатором protected, наследуются в производном классе с тем же самым уровнем доступа.

 **Описание производного класса**:

* Производный класс описывается с использованием ключевого слова class или struct, за которым следует имя класса, а затем двоеточие и ключевое слово public, за которым указывается имя базового класса.

 **Наследование конструкторов**:

* Конструкторы не наследуются автоматически. Они могут вызывать конструкторы базового класса в списке инициализации или через конструктор копирования.

 **Наследование деструкторов**:

* Деструкторы не наследуются автоматически. Они вызываются в обратном порядке, начиная с деструктора производного класса и заканчивая деструктором базового класса.

 **Порядок конструирования объектов производных классов**:

* При создании объекта производного класса сначала вызывается конструктор базового класса, затем конструктор самого производного класса.

 **Порядок уничтожения объектов производных классов**:

* При уничтожении объекта производного класса сначала вызывается деструктор производного класса, а затем деструктор базового класса.

 **Виртуальные функции и позднее связывание**:

* Виртуальные функции позволяют реализовывать полиморфизм, а механизм позднего связывания (также известный как динамическое связывание) обеспечивает вызов подходящей реализации виртуальной функции на основе типа объекта во время выполнения программы.

 **Виртуальные конструкторы и деструкторы**:

* В C++ конструкторы не могут быть виртуальными. Деструкторы могут быть виртуальными, что полезно при работе с полиморфными объектами, где используется оператор delete.

 **Наследуется ли спецификатор virtual**:

* Спецификатор virtual не наследуется.

 **Открытое наследование**:

* Открытое наследование устанавливает отношение "является" между классами, что означает, что производный класс является типом базового класса.

 **Закрытое наследование**:

* Закрытое наследование устанавливает отношение "реализуется в терминах" между классами, что означает, что производный класс реализует интерфейс базового класса.

 **Принцип подстановки**:

* Принцип подстановки (или принцип Лисков) гласит, что объекты производного класса могут быть использованы везде, где используются объекты базового класса, не нарушая функциональность программы.

 **Компонентные данные объекта x**:

* Объект x будет иметь все компонентные данные, которые определены в классах Student, Employee и Teacher, включая age, name, post и stage.

 **Конструкторы без параметров**:

 Student() : age(0), name("") {}

Employee() : Student(), post("") {}

Teacher() : Employee(), stage(0) {}

 **Конструкторы с параметрами**:

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

Employee(int a, const string& n, const string& p) : Student(a, n), post(p) {}

Teacher(int a, const string& n, const string& p, int s) : Employee(a, n, p), stage(s) {}

 **Конструкторы копирования**:

 Student(const Student& s) : age(s.age), name(s.name) {}

Employee(const Employee& e) : Student(e), post(e.post) {}

Teacher(const Teacher& t) : Employee(t), stage(t.stage) {}

 **Операции присваивания**:

Student& operator=(const Student& s) { age = s.age; name = s.name; return \*this; }

Employee& operator=(const Employee& e) { static\_cast<Student&>(\*this) = e; post = e.post; return \*this; }

Teacher& operator=(const Teacher& t) { static\_cast<Employee&>(\*this) = t; stage = t.st