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

ПНИПУ

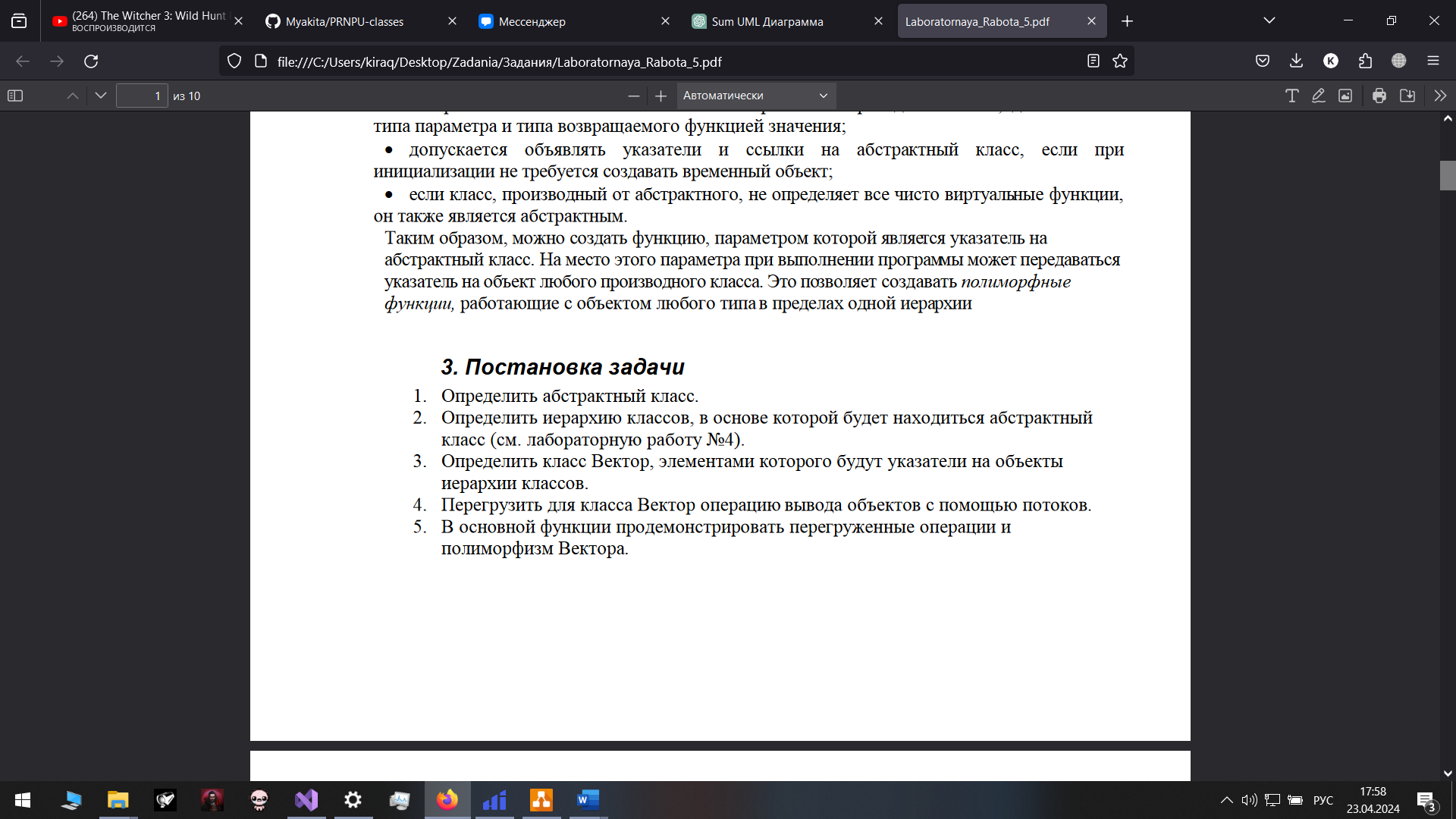
Лабораторная работа  
«Виртуальные функции. Наследование»

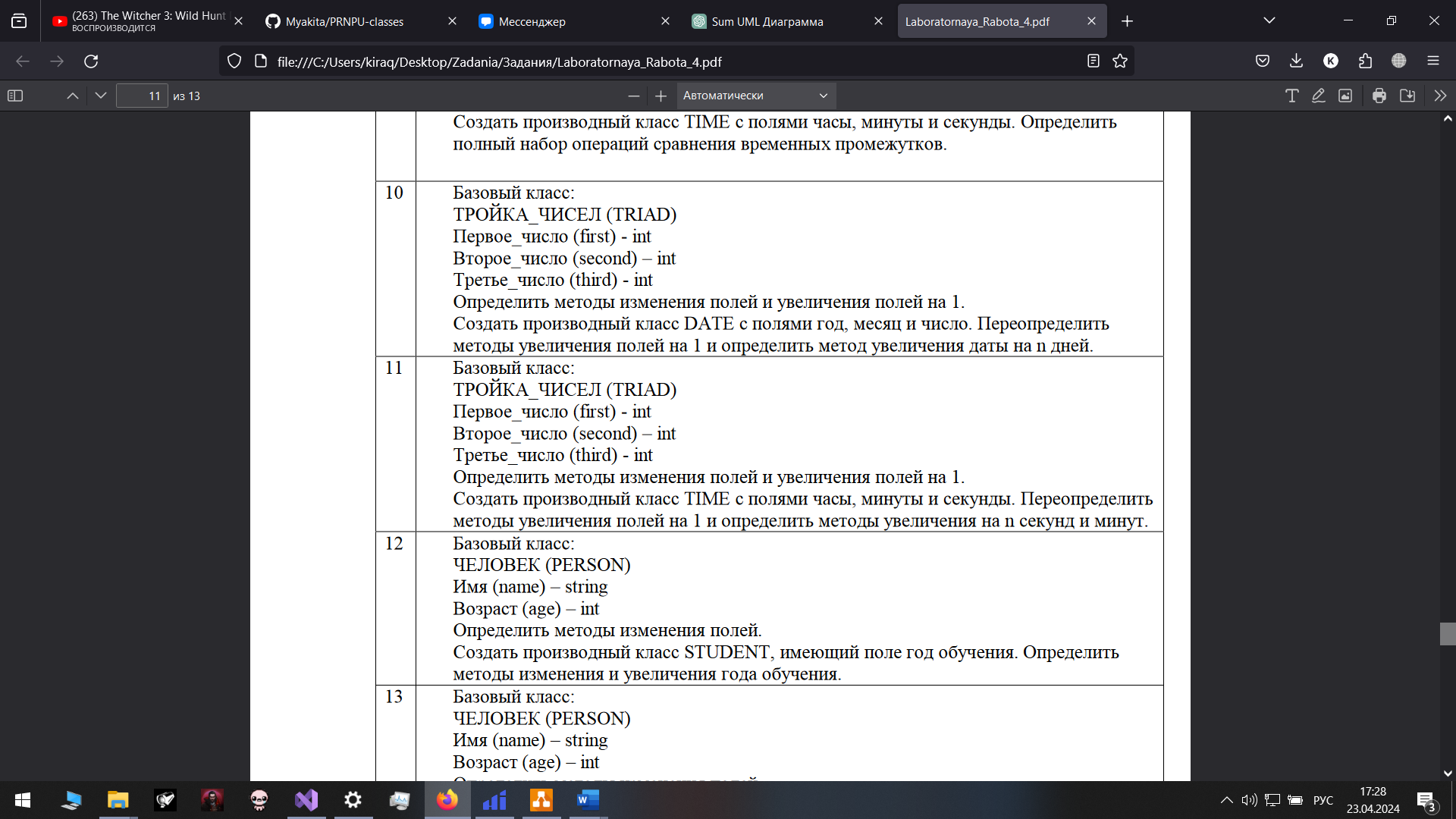
Выполнил:   
студент группы РИС-23-1б   
Мокрушин Никита Дмитриевич

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

Пермь 2024 г.

Постановка задачи:





Анализ задачи:

1. В коде определены три класса: Object, Triad, Time и класс Vector.
2. Класс Object является абстрактным базовым классом с чисто виртуальной функцией output().
3. Класс Triad наследует от класса Object и представляет собой абстракцию тройки целых чисел.
4. Класс Time также наследует от класса Triad и расширяет его функционал, представляя собой абстракцию времени (часов, минут, секунд).
5. В классе Vector реализован простой динамический массив указателей на объекты класса Object.

Код:

#include <iostream>

using namespace std;

class Object {

public:

Object() {};

virtual ~Object() {};

virtual void output() = 0;

};

class Triad : public Object {

friend istream& operator>>(istream& in, Triad& t);

friend ostream& operator<<(ostream& on, const Triad& t);

public:

~Triad() {};

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

Triad(int f, int s, int th) : first(f), second(s), third(th) {}

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

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 th) { third = th; }

Triad& operator=(const Triad& t) {

if (this != &t) {

first = t.first;

second = t.second;

third = t.third;

}

return \*this;

}

void output() override {

cout << "\nfirst: " << first;

cout << "\nsecond: " << second;

cout << "\nthird: " << third;

}

protected:

int first;

int second;

int third;

};

class Time : public Triad {

friend istream& operator>>(istream& in, Time& t);

friend ostream& operator<<(ostream& os, const Time& t);

public:

~Time() {};

Time() : hours(0), minutes(0), seconds(0) {}

Time(const Time& t) : Triad(t), hours(t.hours), minutes(t.minutes), seconds(t.seconds) {}

Time(int h, int m, int s) : Triad(h, m, s), hours(h), minutes(m), seconds(s) {}

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

void setminutes(int m) { minutes = m; }

void setseconds(int s) { seconds = s; }

int gethours() const { return hours; }

int getminutes() const { return minutes; }

int getseconds() const { return seconds; }

Time operator-(const int& minus) const {

return Time(hours, minutes, seconds - minus);

}

Time operator+(const int& plus) const {

return Time(hours, minutes, seconds + plus);

}

void output() override {

cout << "\nhours: " << hours;

cout << "\nminutes: " << minutes;

cout << "\nseconds: " << seconds;

}

void increment(Time& t,int n)

{

t.seconds += n;

if (seconds >= 60)

{

t.minutes += 1;

t.seconds -= 60;

if (t.minutes >= 60)

{

t.hours += 1;

t.minutes -= 60;

if (t.hours >= 24)

{

t.hours -= 24;

}

}

}

}

protected:

int hours;

int minutes;

int seconds;

};

class Vector {

friend ostream& operator<<(ostream& os, const Vector& t);

public:

~Vector() {};

Vector() {

beg = 0;

size = 0;

cur = 0;

};

Vector(int lin) {

beg = new Object \* [lin];

cur = 0;

size = lin;

}

void add(Object\* ptr) {

if (cur < size) {

beg[cur] = ptr;

cur++;

}

}

private:

Object\*\* beg;

int size;

int cur;

};

istream& operator>>(istream& in, Triad& t)

{

cout << "Введите первое: ";

in >> t.first;

cout << "Введите второе: ";

in >> t.second;

cout << "Введите третье: ";

in >> t.third;

return in;

}

istream& operator>>(istream& in, Time& t) {

cout << "Введите часы: ";

in >> t.hours;

cout << "Введите минуты: ";

in >> t.minutes;

cout << "Введите секунды: ";

in >> t.seconds;

return in;

}

ostream& operator<<(ostream& os, const Vector& v)

{

if (v.size == 0)

{

os << "\nВектор пуст";

}

else

{

Object\*\* ptr = v.beg;

for (int i = 0; i < v.cur; i++)

{

(\*ptr)->output();

ptr++;

}

}

return os;

}

ostream& operator<<(ostream& os, const Triad& t)

{

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

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

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

return os;

}

ostream& operator<<(ostream& os, const Time& t)

{

os << "hours: " << t.gethours() << ", minutes: " << t.getminutes() << ", seconds: " << t.getseconds();

return os;

}

int main()

{

setlocale(0, "");

Vector vec(4);

Triad tr;

Time tm;

cout << "Базовый класс: " << endl;

cin >> tr;

cout << tr;

cout << "\nЕще один класс: " << endl;

cin >> tm;

cout << tm;

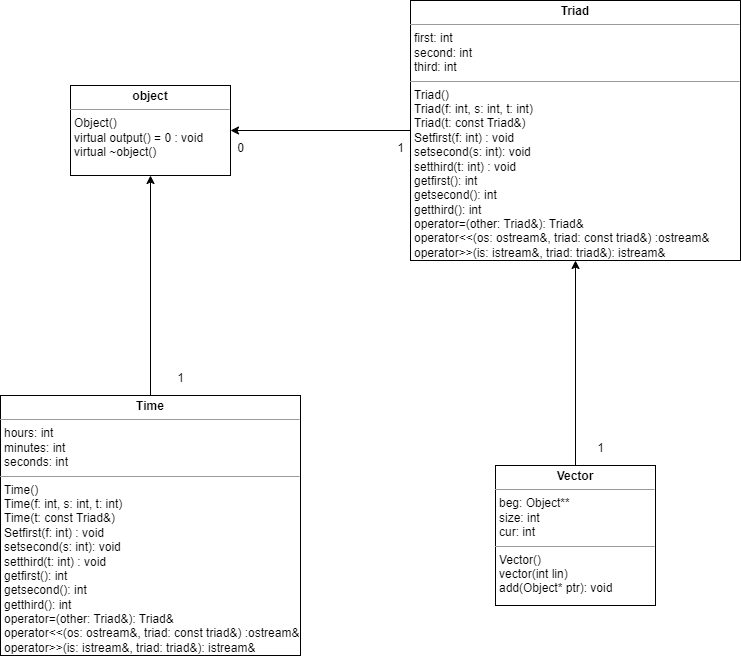
cout << "До инкремента: " << tm << endl;

tm.increment(tm,2);

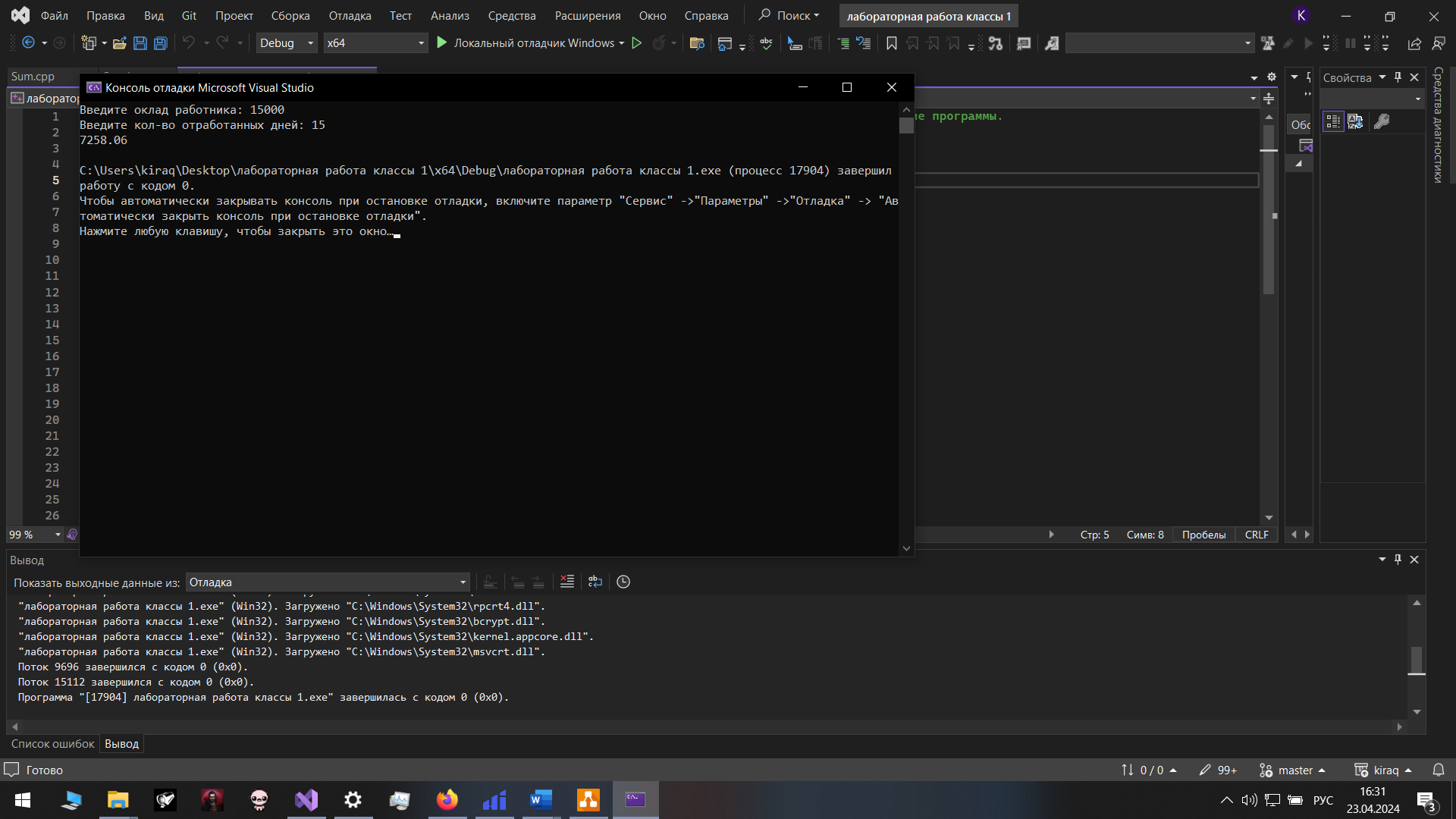
cout << "После инкремента: " << tm << endl;

}

UML Диаграммы:



Вывод:



программа работает корректно и выдаёт требуемый результат.  
Ответы на 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.