Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Информатика»

Тема: Наследование. Виртуальные функции. Полиморфизм.

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Аксёнов Д.О.

Проверил

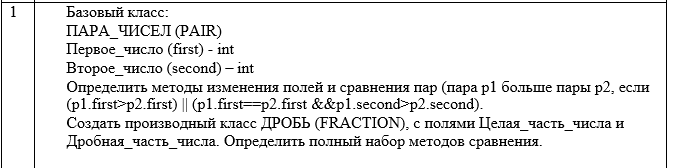
Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

**Постановка задачи (общая и конкретного варианта).**

* + 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.



**Pair.cpp**

#include "Pair.h"

Pair::Pair(int f, int s)

{

first = f;

second = s;

}

Pair::Pair(const Pair& Pair)

{

first = Pair.first;

second = Pair.second;

}

bool Pair::operator>(const Pair& p)

{

return (first > p.first) || (first == p.first && second > p.second);

}

istream& operator >>(istream& in, Pair& p)

{

cout << "First: ";

in >> p.first;

cout << "Second:";

in >> p.second;

return in;

}

ostream& operator<<(ostream& out, const Pair& p)

{

out << p.first << ":" << p.second << "\n";

return out;

}

**Fraction.cpp**

#include "fraction.h"

Fraction::Fraction(int iPart, int fPart) : Pair(iPart, fPart) {}

void Fraction::setIntegerPart(int iPart) {

SetFirst(iPart);

}

void Fraction::setFractionalPart(int fPart) {

SetSecond(fPart);

}

bool Fraction::operator>(const Fraction& f) const {

return (getIntegerPart() > f.getIntegerPart()) || (getIntegerPart() == f.getIntegerPart() && getFractionalPart() > f.getFractionalPart());

}

bool Fraction::operator==(const Fraction& f) const {

return getIntegerPart() == f.getIntegerPart() && getFractionalPart() == f.getFractionalPart();

}

bool Fraction::operator<(const Fraction& f) const {

return f > \*this && !(\*this == f);

}

bool Fraction::operator<=(const Fraction& f) const {

return !(\*this > f);

}

bool Fraction::operator>=(const Fraction& f) const {

return !(f > \*this);

}

**Fraction.h**

#pragma once

#include "Pair.h"

class Fraction : public Pair

{

public:

Fraction(int iPart, int fParct);

int getIntegerPart() const { return GetFirst(); };

int getFractionalPart() const { return GetSecond(); };

void setIntegerPart(int iPart);

void setFractionalPart(int fPart);

bool operator>(const Fraction& f) const;

bool operator==(const Fraction& f) const;

bool operator<(const Fraction& f) const;

bool operator>=(const Fraction& f) const;

bool operator<=(const Fraction& f) const;

};

**Pair.h**

#pragma once

#include <string>

#include <iostream>

using namespace std;

class Pair

{

public:

Pair(int, int);

Pair(const Pair&);

int GetFirst() const { return first; }

int GetSecond() const { return second; }

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

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

bool operator>(const Pair&);

friend istream& operator>>(istream& in, Pair& c);

friend ostream& operator<<(ostream& out, const Pair& c);

protected:

int first;

int second;

}

**Main.cpp**

#include <iostream>

#include "Pair.h"

#include "Fraction.h"

using namespace std;

void main()

{

Pair p1(3, 5);

Pair p2(3, 2);

cout << "Pair 1:\n" << p1 << "Pair 2:\n" << p2;

cout << "p1 > p2: " << boolalpha << (p1 > p2) << endl;

Fraction f1(4, 5);

Fraction f2(3, 7);

cout << "f1 == f2: " << boolalpha << (f1 == f2) << endl;

cout << "f1 > f2: " << boolalpha << (f1 > f2) << endl;

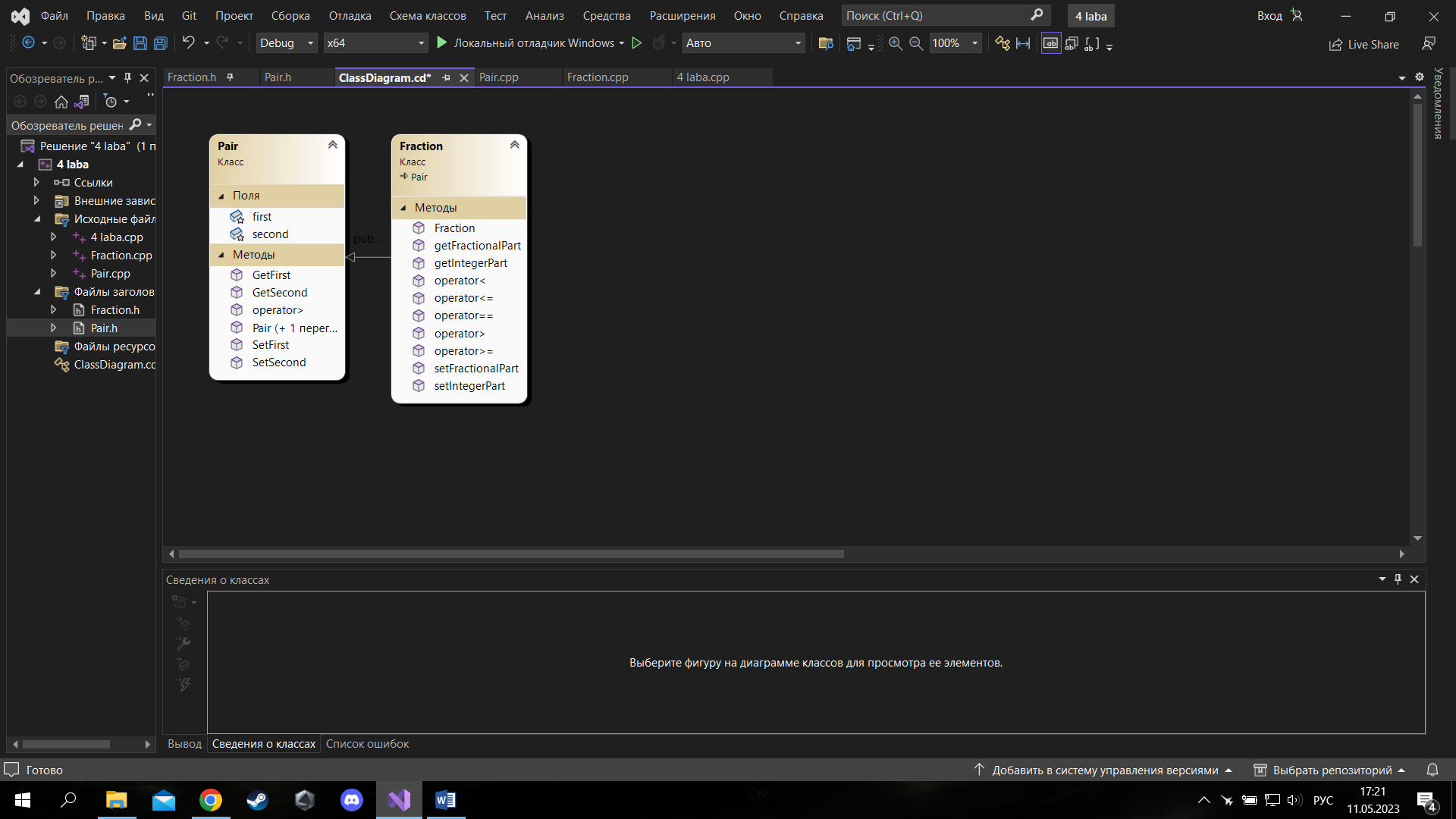
cout << "f1 < f2: " << boolalpha << (f1 < f2) << endl;

cout << "f1 <= f2: " << boolalpha << (f1 <= f2) << endl;

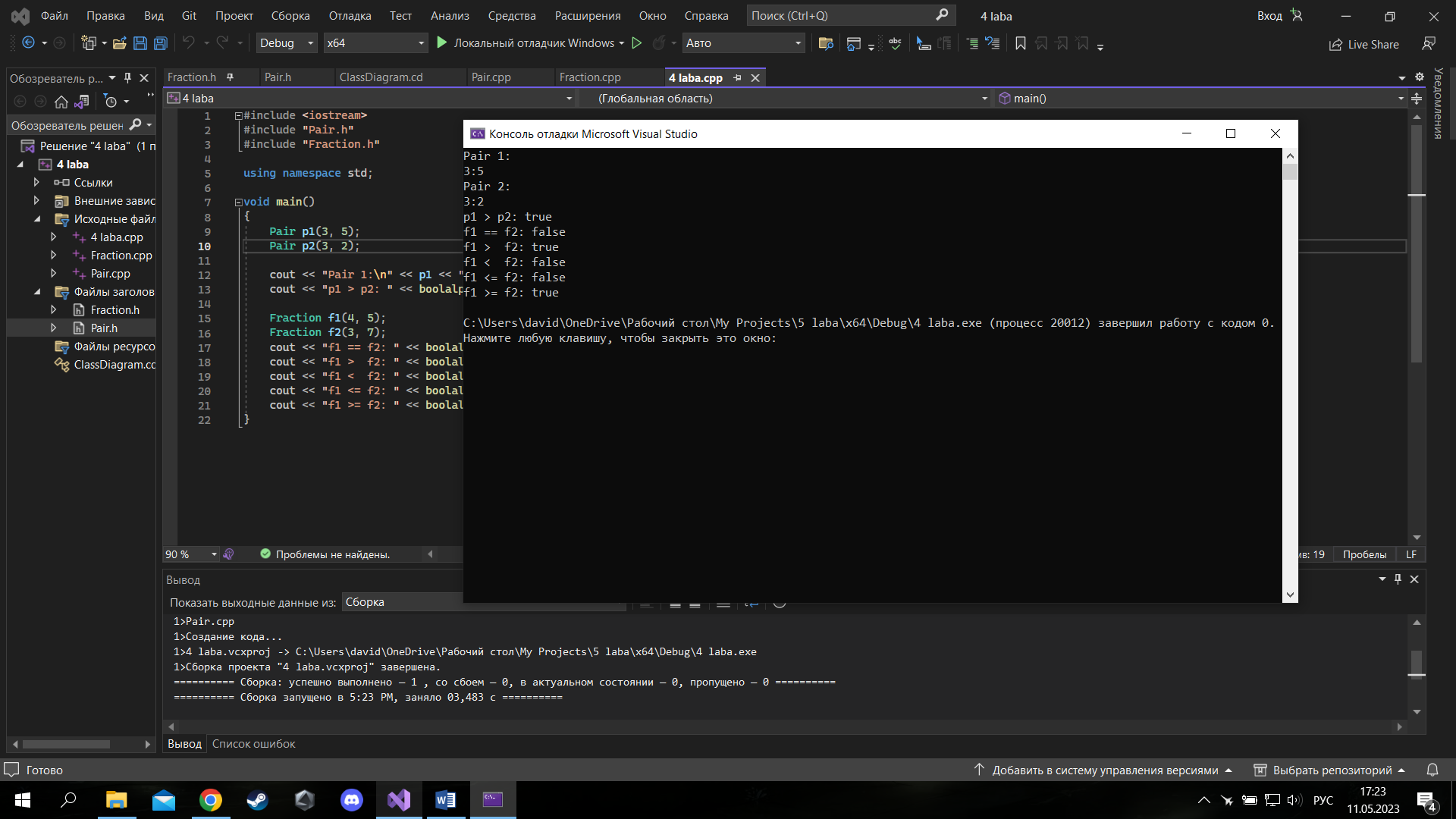
cout << "f1 >= f2: " << boolalpha << (f1 >= f2) << endl;

}

**Uml диаграмма**



**Вывод результатов в консоль.**



**Ответы на контрольные вопросы.**

1. Какой метод называется чисто виртуальным? Чем он отличается от виртуального метода?

Чисто виртуальный метод - виртуальная функция, не имеющая определения в базовом классе. Для страховки от неправильного вызова ее часто объявляют равной нулю: virtual <тип> <имя функции> (<список параметров>) = 0; Чисто виртуальный метод должен переопределяться в производном классе (возможно, опять как чисто виртуальный). Переопределение происходит с помощью спецификатора override (он не обязателен).

2. Какой класс называется абстрактным?

Абстрактный класс - класс, содержащий хотя бы один чисто виртуальный метод.

3. Для чего предназначены абстрактные классы?

Используется как обобщенная концепция, которая используется для создания конкретных производных классов. Создание объектов абстрактного класса невозможно, однако можно использовать указатели и типы на типы абстрактных классов. Абстрактные классы нельзя использовать для приведения типов, но допускается объявление ссылок и указателей, если абстрактный класс имеет наследников. Если производный класс не определяет все чисто виртуальные функции, он также является абстрактным

Пример:

Нужно сделать для класса TextFile базовый класс File, от которого будет унаследован еще один класс RTFFile. Однако, в такой ситуации неизвестно как реализовать метод read() класса File, т.к. класс File не

реализует поведение какого-то конкретного типа файлов, а представляет интерфейс для работы с различными файлами. Для этого используется чисто виртуальная функция.

class File {

virtual string read(int count) = 0;

};

Метод read(...) должен быть определен в классах наследниках. Теперь класс File стал абстрактным, и его экземпляры невозможно создать. Но можно работать через указатель на абстрактный класс с объектами производных классов.

File \*f = new TextFile("text.txt");

//различные действия с файлом text.txt

delete f;

f = new RTFFile("rich\_text.rtf");

//различные действия с файлом rich\_text.rtf

delete f;

4. Что такое полиморфные функции?

При работе с абстрактными классами можно создать функцию, параметром которой будет являться указатель на абстрактный класс. Туда может передаваться указатель на объект любого производного класса, что позволяет создать полиморфные функции, то есть функции, работающие с объектом любого типа в пределах одной иерархии.

class Base {

virtual void work() = 0;

};

class Derived : Base {...};

void print(Base\*a){ cout << “Hello!”}

//в основной функции

Derived \*t = new Derived;

print(t); //хотя в функцию передается базовый файл в качестве аргумента.

5. Чем полиморфизм отличается от принципа подстановки?

Полиморфизм - это возможность классов иметь разную реализацию одного и того же функционала в основном с помощью наследования. Это механизм работы языка программирования.

Принцип подстановки является “правилом хорошего кода". В хорошем коде все методы класса-потомка должны быть применимы к объекту класса родителя.

6. Привести примеры иерархий с использованием абстрактных классов.

class Shape {

public:

virtual double getSquare() const = 0; // площадь фигуры

virtual double getPerimeter() const = 0; // периметр фигуры

};

class Rectangle : public Shape { // класс прямоугольника

public:

Rectangle(double w, double h) : width(w), height(h) { }

double getSquare() const override {

return width \* height;

}

double getPerimeter() const override {

return width \* 2 + height \* 2;

}

private:

double width; // ширина

double height; // высота

};

class Circle : public Shape { // круг

public:

Circle(double r) : radius(r) { }

double getSquare() const override {

return radius \* radius \* 3.14;

}

double getPerimeter() const override {

return 2 \* 3.14 \* radius;

}

private:

double radius; // радиус круга

};