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 ПО ЛАБОРАТОРНОЙ РАБОТЕ**

Дисциплина: «Основы алгоритмизации и программирования»

Тема : "Наследование. Виртуальные функции. Полиморфизм"

Выполнила работу

Студентка группы РИС-22-1Б

Верхоланцева Е. С.

Проверила

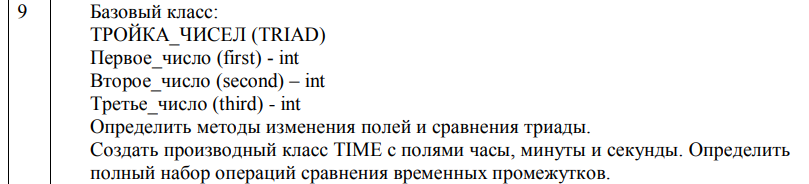
Доцент кафедры ИТАС

Полякова О.А.

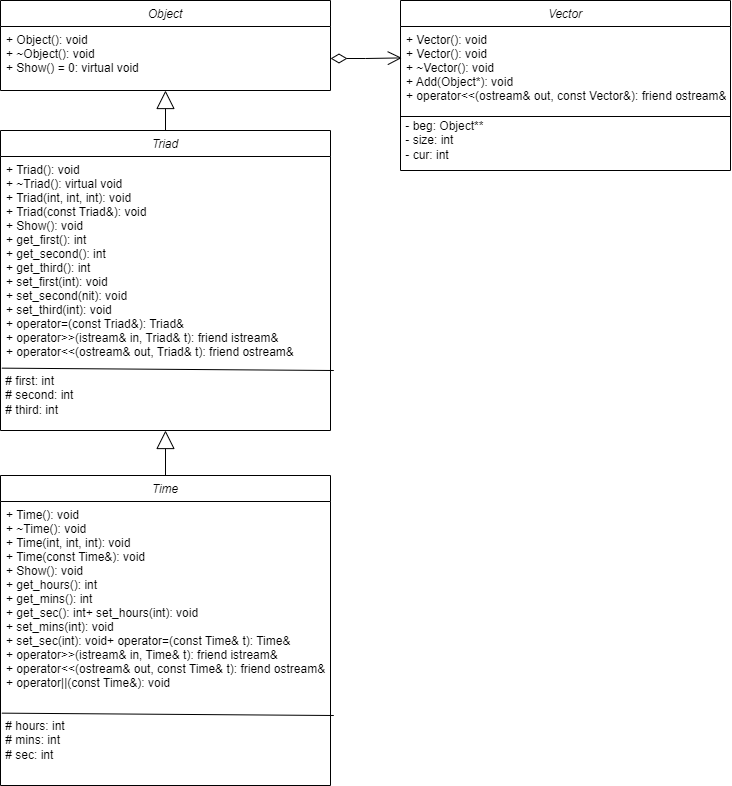
Г. Пермь-2023

**Постановка задачи**

1. Определить абстрактный класс.  
2. Определить иерархию классов, в основе которой будет находиться абстрактный класс (см. лабораторную работу No4).  
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.



**Диаграмма классов**



**Код**

**Описание класса Time**

#pragma once

#include <iostream>

using namespace std;

#include "Triad.h"

class Time :

public Triad

{

public:

Time(void);

~Time(void);

Time(int, int, int);

Time(const Time&);

// селекторы

int get\_hours();

int get\_mins();

int get\_sec();

// модификаторы

void set\_hours(int);

void set\_mins(int);

void set\_sec(int);

void Show();

Time& operator=(const Time& t); // операция присваивания

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

friend ostream& operator<<(ostream& out, const Time& t); // операция вывода

void operator||(const Time&); // операция сравнения

protected:

int hours;

int mins;

int sec;

};

**Определение методов класса Time**

Time::Time(void) :Triad() {

hours = 0;

mins = 0;

sec = 0;

}

Time::~Time(void) {}

Time::Time(int h, int m, int s) :Triad(first, second, third)

{

hours = h;

mins = m;

sec = s;

}

Time::Time(const Time& t)

{

first = t.first;

second = t.second;

third = t.third;

hours = t.hours;

mins = t.mins;

sec = t.sec;

}

// селекторы

int Time::get\_hours() { return hours; }

int Time::get\_mins() { return mins; }

int Time::get\_sec() { return sec; }

// модификаторы

void Time::set\_hours(int h) { hours = h; }

void Time::set\_mins(int m) { mins = m; }

void Time::set\_sec(int s) { sec = s; }

// перегрузка оператора присваивания

Time& Time::operator=(const Time& t)

{

if (&t == this) return \*this;

first = t.first;

second = t.second;

third = t.third;

Time::hours = t.hours;

Time::mins = t.mins;

Time::sec = t.sec;

return \*this;

}

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

{

cout << "Часы: "; in >> t.hours;

cout << "Минуты: "; in >> t.mins;

cout << "Секунды: "; in >> t.sec;

cout << "\n\n";

return in;

}

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

{

out << "\n" << t.hours << " ч " << t.mins << " мин " << t.sec << " сек ";

out << "\n";

return out;

}

void Time::Show()

{

cout << hours << " ч " << mins << " мин " << sec << " сек " << endl;

}

void Time::operator||(const Time& t) // операция сравнения

{

int a = 0;

if (hours > t.hours) {

a = 1;

}

else if (hours < t.hours) { a = 2; }

else {

if (mins > t.mins) { a = 1; }

else if (mins < t.mins) { a = 2; }

else {

if (sec > t.sec) { a = 1; }

else if (sec < t.sec) { a = 2; }

else { a = 3; }

}

}

if (a == 1) { cout << hours << " ч " << mins << " мин " << sec << " сек > " << t.hours << " ч " << t.mins << " мин " << t.sec << " сек " << endl; }

else if (a == 2) { cout << hours << " ч " << mins << " мин " << sec << " сек < " << t.hours << " ч " << t.mins << " мин " << t.sec << " сек " << endl; }

else { cout << hours << " ч " << mins << " мин " << sec << " сек = " << t.hours << " ч " << t.mins << " мин " << t.sec << " сек " << endl; }

}

**Описание класса Triad**

class Triad:

public Object

{

public:

Triad();

virtual ~Triad(void);

Triad(int, int, int);

Triad(const Triad&);

void Show();

// селекторы

int get\_first();

int get\_second();

int get\_third();

// модификаторы

void set\_first(int);

void set\_second(int);

void set\_third(int);

Triad& operator=(const Triad&);

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

friend ostream& operator<<(ostream& out, Triad& t);

protected:

int first;

int second;

int third;

};

**Определение методов класса Triad**

Triad::Triad()

{

first = 0;

second = 0;

third = 0;

cout << "Вызван конструктор без параметров для объекта " << this << endl;

}

Triad::~Triad()

{

cout << "Вызван деструктор для объекта " << this << endl;

}

Triad::Triad(int valueF, int valueS, int valueT)

{

first = valueF;

second = valueS;

third = valueT;

}

Triad::Triad(const Triad& triad)

{

first = triad.first;

second = triad.second;

third = triad.third;

}

void Triad::Show() {

cout << "first = " << first << " second = " << second << " third = " << third << endl;

}

void Triad::set\_first(int f) { first = f; }

void Triad::set\_second(int s) { second = s; }

void Triad::set\_third(int t) { third = t; }

int Triad::get\_first() { return first; }

int Triad::get\_second() { return second; }

int Triad::get\_third() { return third; }

Triad& Triad::operator=(const Triad& t)

{

if (&t == this) return \*this;

first = t.first;

second = t.second;

third = t.third;

return \*this;

}

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

{

cout << "Первое число: "; in >> t.first;

cout << "Второе число: "; in >> t.second;

cout << "Третье число: "; in >> t.third;

cout << "\n";

return in;

}

ostream& operator<<(ostream& out, Triad& t)

{

out << "\nПервое число: " << t.get\_first();

out << "\nВторое число: " << t.get\_second();

out << "\nТретье число: " << t.get\_third();

return out;

}

**Описание класса Object**

class Object

{

public:

Object(void);

~Object(void);

virtual void Show() = 0; // чисто виртуальная функция

};

**Описание класса Vector**

class Vector

{

public:

Vector(void);

Vector(int);

~Vector(void);

void Add(Object\*); // добавление элемента в вектор

friend ostream& operator<<(ostream& out, const Vector&);

private:

Object\*\* beg; // указатель на первый элемент вектора

int size; // размер

int cur; // текущая позиция

};

**Определение методов класса Object**

Vector::Vector(void)

{

beg = 0;

size = 0;

cur = 0;

}

Vector::~Vector(void)

{

if (beg != 0) delete[] beg;

beg = 0;

}

Vector::Vector(int n)

{

beg = new Object \* [n];

cur = 0;

size = n;

}

void Vector::Add(Object\* p)

{

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

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

{

if (v.size == 0) out << "Empty" << endl;

Object\*\* p = v.beg; // указатель на указатель типа Object

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

{

(\*p)->Show(); // вызов метода Show() (позднее связыванеие)

p++; // передвигаем указатель на следующий объект

}

return out;

}

**Определение методов класса Vector**

#include "Vector.h"

#include <iostream>

using namespace std;

Vector::Vector(void)

{

beg = 0;

size = 0;

cur = 0;

}

Vector::~Vector(void)

{

if (beg != 0) delete[] beg;

beg = 0;

}

Vector::Vector(int n)

{

beg = new Object \* [n];

cur = 0;

size = n;

}

void Vector::Add(Object\* p)

{

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

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

{

if (v.size == 0) out << "Empty" << endl;

Object\*\* p = v.beg; // указатель на указатель типа Object

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

{

(\*p)->Show(); // вызов метода Show() (позднее связыванеие)

p++; // передвигаем указатель на следующий объект

}

return out;

}

**Ответы на контрольные вопросы**

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

Чисто виртуальная функция (pure virtual function) – **виртуальная функция, не имеющая опеределения в базовом классе**.

Для страховки от неправильного вызова ее часто объявляют равной нулю: virtual <тип> <имя функции> (<список параметров>) = 0;

Основное различие между виртуальной функцией и чисто виртуальной функцией является то, что виртуальная функция является функцией в базовом классе, который объявлен с использованием виртуального ключевого слова в то время как чистая виртуальная функция является виртуальной функцией в базовом классе без определения функции.



2. Какой класс называется абстрактным?

***Абстрактный класс*** в объектно-ориентированном программировании — базовый класс, который не предполагает создания экземпляров.

3. Для чего предназначены абстрактные классы?

Абстрактные классы предназначены для представления общих понятий, которые предполагается конкретизировать в производных классах. Абстрактный класс может использоваться только в качестве базового для других классов — объекты абстрактного класса создавать нельзя, поскольку прямой или косвенный вызов чисто виртуального метода приводит к ошибке при выполнении.

4. Что такое полиморфные функции?

*Полиморфизм* – это свойство программного кода изменять свое поведение в зависимости от ситуации, возникающей при выполнении программы. В контексте реализации *полиморфизм* – это технология вызова виртуальных функций, реализуемых в иерархически связанных классах.

 Полиморцная функция - это специальным образом оформленная функция, которая может быть в так называемом полиморфном состоянии – состоянии, при котором вызов нужной функции из набора виртуальных формируется на этапе *позднего связывания*. Понятие позднее связывание означает, что код вызова нужной функции формируется при выполнении программы.

5. Чем полиморфизм отличается от принципа подстановки?

При полиморфизме ф-ия может изменять свой функционал.

6. Привести примеры иерархий с использованием абстрактных классов.

class Figure

{

public:

    virtual double getSquare() =0;

    virtual double getPerimeter() =0;

    virtual void showFigureType()=0;

};

class Rectangle : public Figure

{

private:

    double width;

    double height;

public:

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

    {

    }

    double getSquare() override

    {

        return width \* height;

    }

    double getPerimeter() override

    {

        return width \* 2 + height \* 2;

    }

    void showFigureType()

    {

        std::cout << "Rectangle" << std::endl;

    }

};

class Circle : public Figure

{

private:

    double radius;

public:

    Circle(double r) : radius(r)

    {

    }

    double getSquare() override

    {

        return radius \* radius \* 3.14;

    }

    double getPerimeter() override

    {

        return 2 \* 3.14 \* radius;

    }

    void showFigureType()

    {

        std::cout << "Circle" << std::endl;

    }

};

int main()

{

    Rectangle rect(30, 50);

    Circle circle(30);

    std::cout << "Rectangle square: " << rect.getSquare() << std::endl;

    std::cout << "Circle square: " << circle.getSquare() << std::endl;

    return 0;

}

7. Привести примеры полиморфных функций.

class Abstract{ //Абстрактный класс

public:virtual void print\_msg()=0;

};

8. В каких случаях используется механизм позднего связывания?

Позднее связывание означает, что объект связывается с вызовом функции только во время исполнения программы, а не раньше. Позднее связывание достигается в С++ с помощью использования виртуальных функций и производных классов.