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: «Основы алгоритмизации и программирования»

Тема: «Наследование. Виртуальные функции. Полиморфизм.»

Выполнил работу

Студент группы РИС-23-3Б

Шуракова А.А.

Проверил

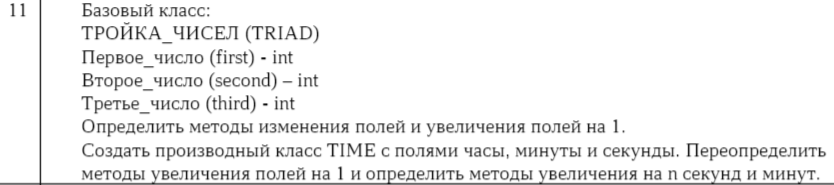
Доцент кафедры ИТАС

Полякова О.А.

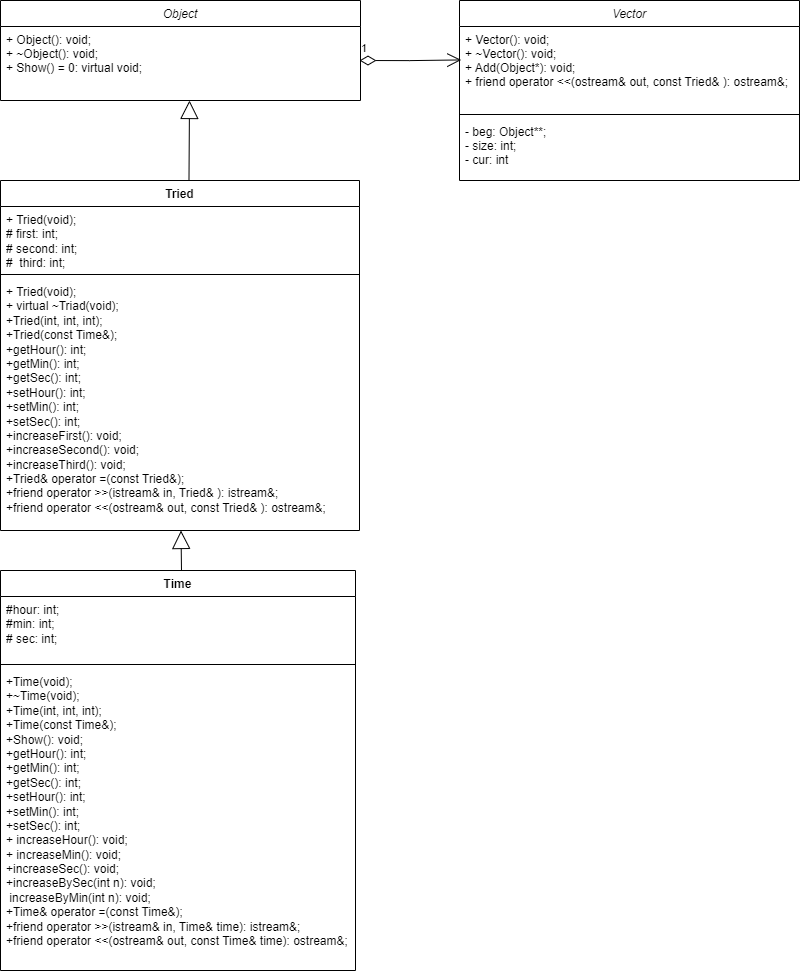
Г. Пермь-2024

**Постановка задачи:**

* Определить абстрактный класс.
* Определить иерархию классов, в основе которой будет находиться абстрактный класс (см. лабораторную работу №4).
* Определить класс Вектор, элементами которого будут указатели на объекты иерархии классов.
* Перегрузить для класса Вектор операцию вывода объектов с помощью потоков.
* В основной функции продемонстрировать перегруженные операции и полиморфизм Вектора.



**Uml диаграмма:**

****

**Код программы:**

Triad.h

#pragma once

#include "Object.h"

#include <iostream>

using namespace std;

class Triad :

public Object {

public:

Triad(void);

public:

virtual ~Triad(void);

void Show() {} //функция для просмотра атрибутов класса с помощью указателя

Triad(int, int, int);

Triad(const Triad&);

int getFirst() { return first; }

int getSecond() { return second; }

int getThird() { return third; }

void increaseFirst();

void increaseSecond();

void increaseThird();

void setFirst(int);

void setSecond(int);

void setThird(int);

Triad& operator =(const Triad&);

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

friend ostream& operator <<(ostream& out, const Triad& T);

protected:

int first, second, third;

};

Triad.cpp

#include "Triad.h"

#include <iostream>

Triad::Triad(void) {

first = 0;

second = 0;

third = 0;

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

}

Triad::~Triad(void) {

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

}

Triad::Triad(int F, int S, int T) {

first = F;

second = S;

third = T;

}

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::increaseFirst() { first++; }

void Triad::increaseSecond() { second++; }

void Triad::increaseThird() { third++; }

void Triad::setFirst(int F) {

first = F;

}

void Triad::setSecond(int S) {

second = S;

}

void Triad::setThird(int T) {

third = T;

}

Triad& Triad::operator =(const Triad& triad) {

if (&triad == this) {

return \*this;

}

first = triad.first;

second = triad.second;

third = triad.third;

return \*this;

}

istream& operator >>(istream& in, Triad& T) {

cout << "\nFirst: "; in >> T.first;

cout << "\nSecond: "; in >> T.second;

cout << "\nThird: "; in >> T.third;

return in;

}

ostream& operator <<(ostream& out, const Triad& T) {

out << "\nFIRST: " << T.first;

out << "\nSECOND: " << T.second;

out << "\nTHIRD: " << T.third;

return out;

}

Time.h

#pragma once

#include <iostream>

#include "Triad.h"

using namespace std;

class Time :

public Triad {

public:

Time(void);

public:

~Time(void);

Time(int, int, int);

Time(const Time&);

void Show();

int getHour() { return hour; }

int getMin() { return min; }

int getSec() { return sec; }

void increaseHour();

void increaseMin();

void increaseSec();

void increaseBySec(int n);

void increaseByMin(int n);

void setHour(int);

void setMin(int);

void setSec(int);

Time& operator =(const Time&);

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

friend ostream& operator <<(ostream& out, const Time& time);

protected:

int hour, min, sec;

};

Time.cpp

#include <iostream>

#include "Time.h"

using namespace std;

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

hour = 0;

min = 0;

sec = 0;

}

Time::~Time(void) {}

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

{

hour = h;

min = m;

sec = s;

}

Time::Time(const Time& t)

{

first = t.first;

second = t.second;

third = t.third;

hour = t.hour;

min = t.min;

sec = t.sec;

}

void Time::increaseHour() {

hour++;

if (hour > 23) {

hour = 0;

}

}

void Time::increaseMin() {

min++;

if (min > 59) {

min = 0;

increaseHour();

}

}

void Time::increaseSec() {

sec++;

if (sec > 59) {

sec = 0;

increaseMin();

}

}

void Time::increaseBySec(int n) {

sec += n;

while (sec > 59) {

sec -= 60;

increaseMin();

}

}

void Time::increaseByMin(int n) {

min += n;

while (min > 59) {

min -= 60;

increaseHour();

}

}

// модификаторы

void Time::setHour(int F) {

first = F;

}

void Time::setMin(int S) {

second = S;

}

void Time::setSec(int T) {

third = T;

}

// перегрузка оператора присваивания

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

{

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

first = t.first;

second = t.second;

third = t.third;

Time::hour = t.hour;

Time::min = t.min;

Time::sec = t.sec;

return \*this;

}

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

{

cout << "Часы: "; in >> t.hour;

cout << "Минуты: "; in >> t.min;

cout << "Секунды: "; in >> t.sec;

cout << "\n\n";

return in;

}

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

{

out << "\n" << t.hour << " ч " << t.min << " мин " << t.sec << " сек ";

out << "\n";

return out;

}

void Time::Show()

{

cout << hour << " ч " << min << " мин " << sec << " сек " << endl;

}

Object.h

#pragma once

class Object

{

public:

Object(void);

public:

~Object(void);

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

};

Object.cpp

#include <iostream>

#include "Object.h"

using namespace std;

Object::Object() {}

Object::~Object() {}

Vector.h

#pragma once

#include <iostream>

#include "Object.h"

using namespace std;

class Vector {

public:

Vector(void);

Vector(int);

public:

~Vector(void);

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

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

private:

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

int size, cur; //размер, текущая позиция

};

Vector.cpp

#include <iostream>

#include "Vector.h"

using namespace std;

Vector::Vector() {

ptr = 0;

size = 0;

cur = 0;

}

Vector::~Vector() {

if (ptr != 0) {

delete[] ptr;

}

ptr = 0;

}

Vector::Vector(int n) {

ptr = new Object \* [n];

cur = 0;

size = n;

}

void Vector::Add(Object\* p) {

if (cur < size) {

ptr[cur] = p;

cur++;

}

}

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

if (v.size == 0) {

out << "EMPTY" << endl;

}

Object\*\* p = v.ptr; // 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;

}

Lab\_main.cpp

#include <iostream>

#include "Time.h"

#include "Object.h"

#include "Triad.h"

#include "Vector.h"

using namespace std;

int main()

{

system("chcp 1251");

Time time;

cin >> time;

cout << time << endl;

int n;

cout << "На сколько секунд увеличить ";

cin >> n;

time.increaseBySec(n);

cout << "После увеличения на " << n << " секунд : ";

cout << time << endl;

cout << "На сколько минут увеличить ";

cin >> n;

time.increaseByMin(n);

cout << "После увеличения на " << n << " минут : ";

cout << time << endl;

cout << endl;

cout << "переопределение методов увеличения на 1" << endl;

Triad f;

cin >> f;

f.increaseFirst();

f.increaseSecond();

f.increaseThird();

cout << f<<endl;

Vector v(5);

Object\* p = &f;

v.Add(p);

p = &time;

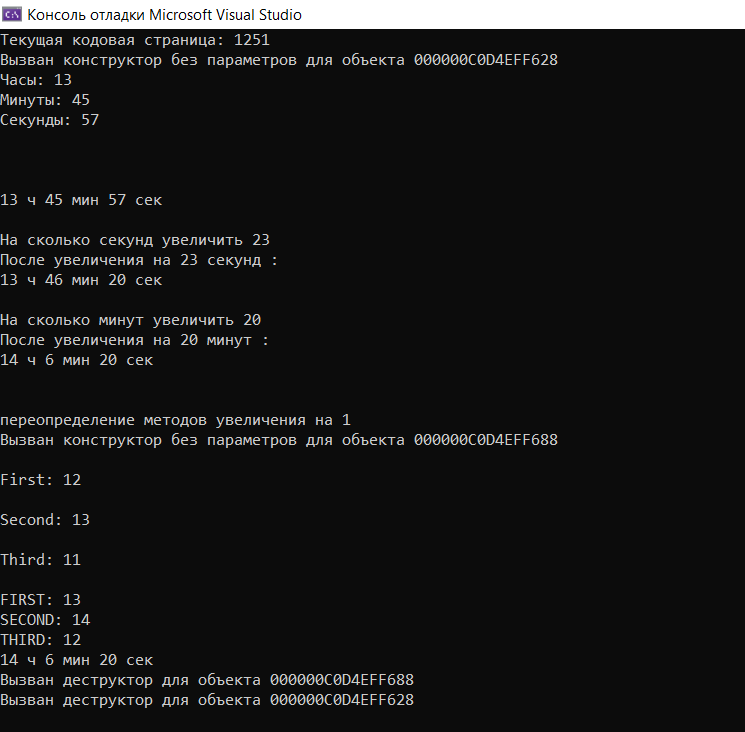
v.Add(p);

cout << v;

return 0;

}

**Результат работы программы:**

****

**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.

Полиморфная 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 функция 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()

{

cout << "Rectangle" << 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()

{

cout << "Circle" << endl;

}

};

int main()

{

Rectangle rect(30, 50);

Circle circle(30);

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

cout << "Circle square: " << circle.getSquare() << 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.

**Ссылка на Git:**

https://github.com/Ananasic07/labs\_2sem/tree/main/Класс%205