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**

Тема: «Наследование. Виртуальные функции. Полиморфизм»

Выполнил работу

Студент группы РИС-24-1Б

Конькова С. С.

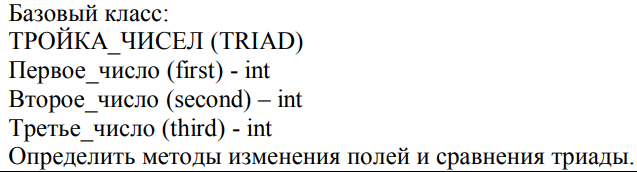
Проверил

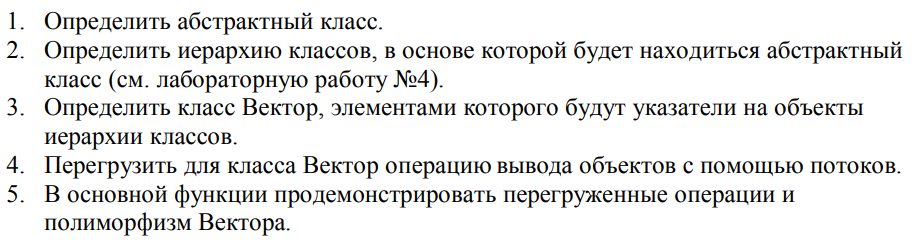
Доцент кафедры ИТАС

Полякова О.А.

Пермь 2025

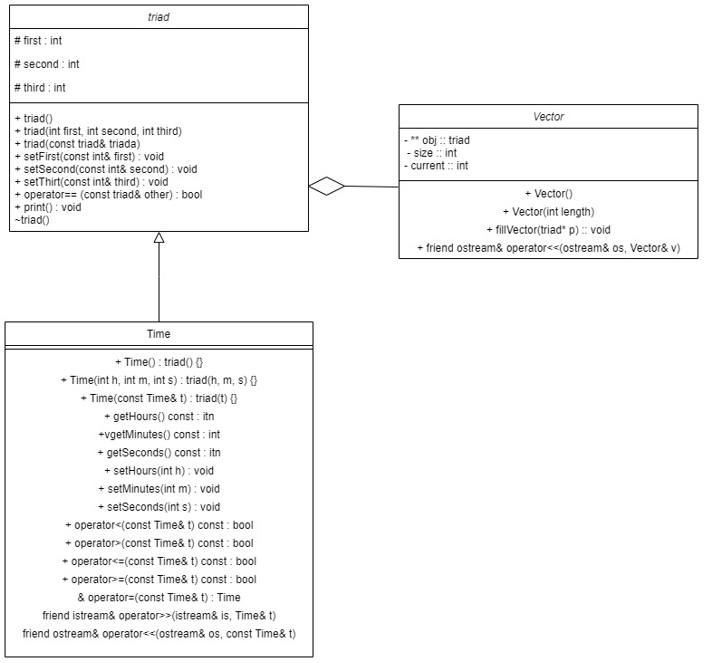
Постановка задачи.





**Анализ**

1. Класс triad (триада) - Базовый класс для хранения трех целых чисел
2. Класс Time (наследник triad) - Специализированный класс для работы со временем
3. Класс Vector - Контейнер для хранения указателей на объекты triad
4. Создается вектор vec размером 2
5. Создаются объекты tr (triad) и tim (Time)
6. Пользователь вводит значения для триады (по одному числу)
7. Пользователь вводит время (часы, минуты, секунды)
8. Указатели на tr и tim добавляются в вектор
9. Содержимое вектора выводится на экран

UML диаграмма.

Код программы.

Header.h

#pragma once

#pragma once

#include<iostream>

using namespace std;

class triad {

protected:

int first, second, third;

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;

}

int getFirst() const;

int getSecond() const;

int getThird() const;

void setFirst(int f);

void setSecond(int s);

void setThird(int t);

void print();

bool operator==(const triad& t) const;

bool operator!=(const triad& t) const;

~triad() {}

};

class Time : public triad

{

public:

Time() : triad() {}

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

Time(const Time& t) : triad(t) {}

int getHours() const;

int getMinutes() const;

int getSeconds() const;

void setHours(int h);

void setMinutes(int m);

void setSeconds(int s);

bool operator<(const Time& t) const;

bool operator>(const Time& t) const;

bool operator<=(const Time& t) const;

bool operator>=(const Time& t) const;

Time& operator=(const Time& t);

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

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

};

class Vector

{

private:

triad\*\* obj;

int size;

int current;

public:

Vector()

{

obj = 0;

size = 0;

current = 0;

}

Vector(int length)

{

obj = new triad \* [length];

current = 0;

size = length;

}

void fillVector(triad\* p);

friend ostream& operator<<(ostream& os, Vector& v);

};

Header.cpp

#include"Header.h"

int triad::getFirst() const

{

return first;

}

int triad::getSecond() const

{

return second;

}

int triad::getThird() const

{

return third;

}

void triad::setFirst(int f)

{

first = f;

}

void triad::setSecond(int s)

{

second = s;

}

void triad::setThird(int t)

{

third = t;

}

void triad::print()

{

cout << "(" << first << ", " << second << ", " << third << ")";

}

bool triad:: operator==(const triad& t) const

{

return (first == t.first && second == t.second && third == t.third);

}

bool triad:: operator!=(const triad& t) const

{

return !(operator==(t));

}

int Time::getHours() const

{

return first;

}

int Time::getMinutes() const

{

return second;

}

int Time::getSeconds() const

{

return third;

}

void Time::setHours(int h)

{

first = h;

}

void Time::setMinutes(int m)

{

second = m;

}

void Time::setSeconds(int s)

{

third = s;

}

bool Time:: operator<(const Time& t) const

{

if (first < t.first) return true;

else if (first == t.first && second < t.second) return true;

else if (first == t.first && second == t.second && third < t.third) return true;

else return false;

}

bool Time:: operator>(const Time& t) const

{

return !(operator<(t) || operator==(t));

}

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

{

return operator<(t) || operator==(t);

}

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

{

return !operator<(t);

}

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

{

if (this != &t)

{

first = t.first;

second = t.second;

third = t.third;

}

return \*this;

}

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

{

cout << endl << "Введите часы: ";

is >> t.first;

cout << "Введите минуты: ";

is >> t.second;

cout << "Введите секунды: ";

is >> t.third;

return is;

}

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

{

os << t.first << ":" << t.second << ":" << t.third;

return os;

}

void Vector:: fillVector(triad\* p)

{

if (current < size)

{

obj[current] = p;

++current;

}

}

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

{

if (v.size == 0)

cout << "\nВектор пуст";

else

{

triad\*\* p = v.obj;

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

{

(\*p)->print();

++p;

}

}

return os;

}

Sourse.cpp

#include"Header.h"

int main()

{

system("chcp 1251 >> null");

Vector vec(2);

triad tr;

Time tim;

int x, y, z;

cout << "Введите триаду:\n";

cout << "Введите первое число триады: ";

cin >> x;

tr.setFirst(x);

cout << "Введите второе число триады: ";

cin >> y;

tr.setSecond(y);

cout << "Введите третье число триады: ";

cin >> z;

tr.setThird(z);

cout << "Триада: ";

tr.print();

cout << "\n\nВведите время:\n";

cin >> tim;

cout << tim;

triad\* pr = &tr;

vec.fillVector(pr);

pr = &tim;

vec.fillVector(pr);

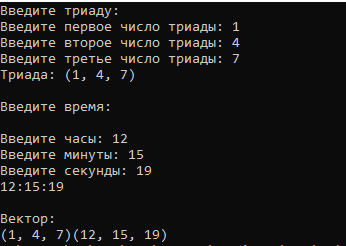
cout << "\n\nВектор:\n";

cout << vec;

return 0;

}

Вывод.



Контрольные вопросы

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

Чисто виртуальным методом называется метод, который не имеет реализации в базовом классе и должен быть переопределен в производных классах. Он отличается от виртуального метода тем, что виртуальный метод имеет реализацию в базовом классе, но может быть переопределен в производных классах.

2. Какой класс называется абстрактным?

Абстрактным классом называется класс, который содержит хотя бы один чисто виртуальный метод.

3. Для чего предназначены абстрактные классы?

Абстрактные классы предназначены для создания иерархий классов, где базовый класс определяет общие свойства и методы, а производные классы реализуют специфические детали. Они также позволяют определить интерфейсы для классов, которые будут реализованы в производных классах.

4. Что такое полиморфные функции?

Полиморфные функции - это функции, которые могут принимать объекты разных классов и вызывать соответствующие им методы.

5. Чем полиморфизм отличается от принципа подстановки?

Полиморфизм - это возможность использовать объекты разных классов через общий интерфейс (например, через абстрактный класс). Принцип подстановки - это принцип, по которому объекты производных классов могут использоваться вместо объектов базового класса без изменения поведения программы.

6. Привести примеры иерархий с использованием абстрактных классов.

Примеры иерархий с использованием абстрактных классов:

- Фигуры: абстрактный класс Figure, производные классы Circle, Square, Triangle.

- Животные: абстрактный класс Animal, производные классы Cat, Dog, Bird.

7. Привести примеры полиморфных функций.

Примеры полиморфных функций:

- Функция draw(), которая принимает объекты разных классов фигур и рисует их на экране.

- Функция makeSound(), которая принимает объекты разных классов животных и воспроизводит соответствующий звук.

8. В каких случаях используется механизм позднего связывания?

Механизм позднего связывания используется в случаях, когда необходимо вызывать методы производных классов через указатель на базовый класс. Это позволяет использовать объекты производных классов через общий интерфейс, сохраняя при этом их специфические свойства и методы.