Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы» направление подготовки: 09.03.04 – «Программная инженерия»

# Лабораторная работа " Наследование. Виртуальные функции. Полиморфизм."

Выполнил студент гр. РИС-24-3б

Жиряков Леонид Антонович

Проверил:

Доцент кафедры ИТАС   
Ольга Андреевна Полякова

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (оценка) (подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

г. Пермь, 2024

Вариант 8.

Постановка задачи:

Общая:

1. Определить абстрактный класс.

2. Определить иерархию классов, в основе которой будет находиться абстрактный класс.

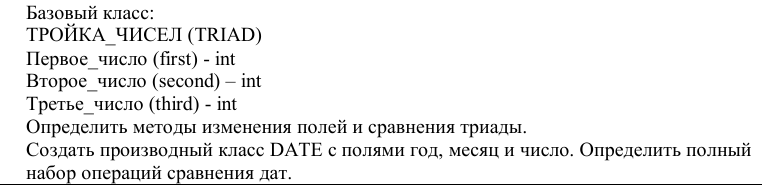
3. Определить класс Вектор, элементами которого будут указатели на объекты иерархии классов.

4. Перегрузить для класса Вектор операцию вывода объектов с помощью

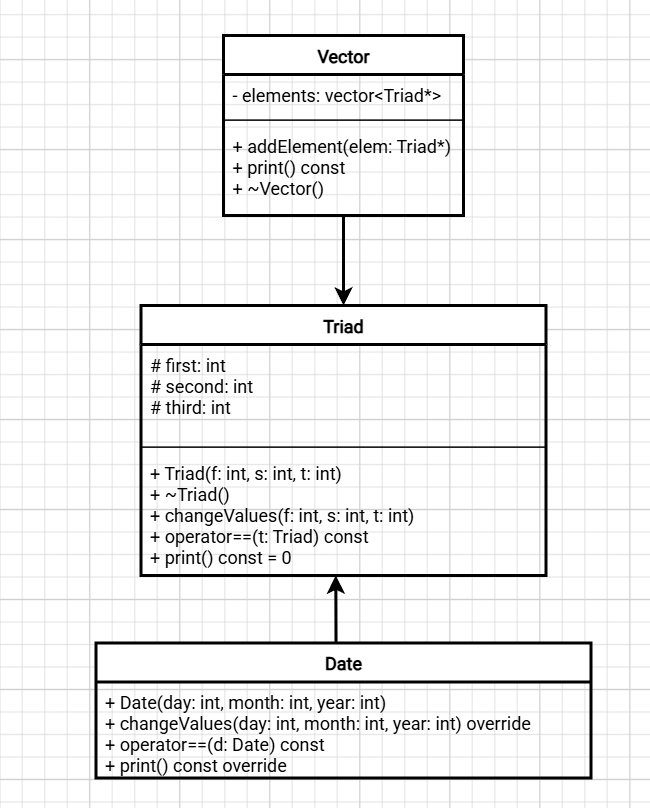
потоков.

5. В основной функции продемонстрировать перегруженные операции и полиморфизм Вектора.

Персональная:



UML – диаграмма:



Программная реализация

**Заголовочные файлы**

#pragma once

using namespace std;

class Triad

{

protected:

int first;

int second;

int third;

public:

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

virtual ~Triad() {}

virtual void changeValues(int f, int s, int t)

{

first = f; second = s; third = t;

}

virtual bool operator==(const Triad& t) const

{

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

}

virtual void print() const = 0; // виртуальная функция

};

#pragma once

#include <vector>

#include <iostream>

#include "Triada.h"

class Vector

{

private:

vector<Triad\*> elements;

public:

void addElement(Triad\* elem)

{

elements.push\_back(elem);

}

void print() const

{

for (const auto& elem : elements)

{

elem->print();

}

}

~Vector()

{

for (auto& elem : elements)

{

delete elem; // освобождение памяти

}

}

};

#pragma once

#include "Triada.h"

#include <iostream>

class Date : public Triad

{

public:

Date(int day, int month, int year) : Triad(day, month, year) {}

void changeValues(int day, int month, int year) override

{

first = day; second = month; third = year;

}

bool operator==(const Date& d) const

{

return Triad::operator==(d);

}

void print() const override

{

cout << "Date: " << first << "/" << second << "/" << third << std::endl; // ДД/ММ/ГГГГ

}

};

**Функция main**

#include <iostream>

#include "Date.h"

#include "Vector.h"

int main()

{

setlocale(LC\_ALL, "RU");

Vector vector;

Date\* date1 = new Date(22, 4, 1870);

Date\* date2 = new Date(22, 4, 1870);

vector.addElement(date1);

vector.addElement(date2);

vector.print(); // демонстрация полиморфизма

if (\*date1 == \*date2)

{

cout << "Даты совпадают." << endl;

}

else

{

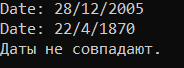
cout << "Даты не совпадают." << endl;

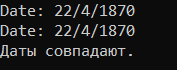
}

return 0;

}

Результат работы:





Контрольные вопросы:

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

2. Какой класс называется абстрактным?

3. Для чего предназначены абстрактные классы?

4. Что такое полиморфные функции?

5. Чем полиморфизм отличается от принципа подстановки?

6. Привести примеры иерархий с использованием абстрактных классов.

7. Привести примеры полиморфных функций.

8. В каких случаях используется механизм позднего связывания?

Ответы:

1. **Чисто виртуальный метод** — это метод, который объявляется с использованием = 0 вместо тела, например: virtual void f(int) = 0;. Он отличается от виртуального метода тем, что чисто виртуальный метод не имеет реализации в базовом классе и должен быть переопределен в производных классах, в то время как виртуальный метод может иметь реализацию.
2. **Абстрактным классом** называется класс, содержащий хотя бы один чисто виртуальный метод. Объекты абстрактного класса создавать нельзя.
3. **Абстрактные классы** предназначены для представления общих понятий, которые предполагается конкретизировать в производных классах. Они служат основой для создания иерархий классов.
4. **Полиморфные функции** — это функции, которые могут работать с объектами разных производных классов, основанных на абстрактном классе, благодаря механизму позднего связывания.
5. **Полиморфизм** — это способность объектов разных классов обрабатывать одинаковые сообщения, а **принцип подстановки** (или подтипирование) — это возможность использовать производный класс вместо базового. Полиморфизм позволяет вызывать методы на базовом классе, а принцип подстановки гарантирует, что производные классы могут быть использованы вместо базового без изменения поведения программы.
6. Примеры иерархий с использованием абстрактных классов:
   * **Транспортные средства**: абстрактный класс ТранспортноеСредство с чисто виртуальными методами двигаться() и остановиться(), производные классы: Автомобиль, Велосипед, Самолёт.
   * **Фигура**: абстрактный класс Фигура с чисто виртуальным методом вычислитьПлощадь(), производные классы: Квадрат, Круг, Треугольник.
7. Примеры полиморфных функций:
   * Функция void рисовать(Фигура\* фигура), которая принимает указатель на абстрактный класс Фигура и вызывает метод вычислитьПлощадь(). В зависимости от типа переданной фигуры, будет вызван соответствующий метод.
   * Функция void перемещать(ТранспортноеСредство\* транспорт), которая вызывает метод двигаться() для любого транспортного средства.
8. **Механизм позднего связывания** используется в случаях, когда необходимо определить, какой метод будет вызван в момент выполнения программы, а не на этапе компиляции. Это позволяет динамически вызывать методы производных классов, что особенно полезно в полиморфных ситуациях. Например, при работе с коллекциями объектов разных типов, основанных на одном абстрактном классе.

Ссылка:

**https://github.com/LeonidZhir/-**