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-22-2Б

Третьяков Н. А.

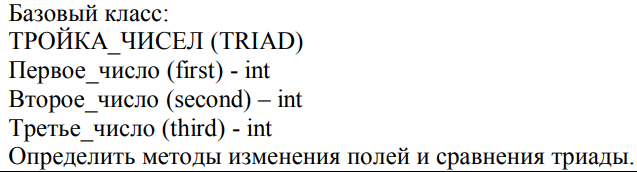
Проверил

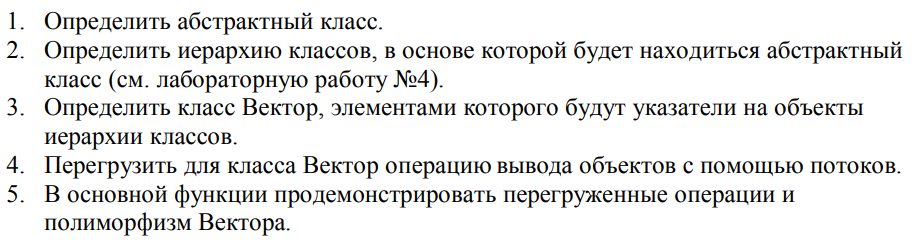
Доцент кафедры ИТАС

Полякова О.А.

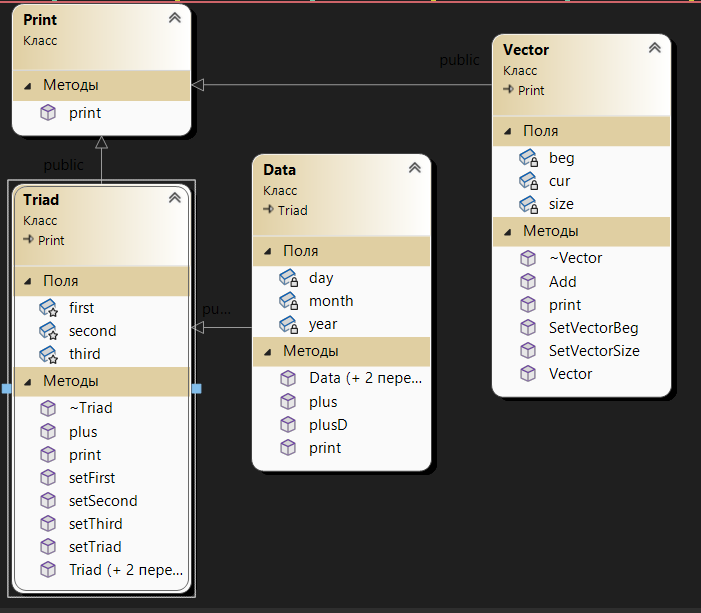
Г. Пермь-2023

Постановка задачи.





UML диаграмма.



Код программы.

Print.h

#pragma once

#include <iostream>

using namespace std;

class Print

{

public:

virtual void print() = 0;

};

Vector.h

#pragma once

#include <iostream>

#include "Print.h"

using namespace std;

class Vector:public Print

{

Print\*\* beg;

int size;

int cur;

public:

Vector();

void print() override;

void SetVectorSize(int);

void SetVectorBeg();

void Add(Print\*);

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

~Vector() {};

};

Vector.cpp

#include <iostream>

#include "Print.h"

#include "Vector.h"

using namespace std;

Vector::Vector()

{

cur = 0;

size = 0;

beg = 0;

}

void Vector::print()

{

cout << " Vector " << endl;

}

void Vector::SetVectorSize(int a)

{

size = a;

}

void Vector::SetVectorBeg()

{

beg = new Print \* [size];

}

void Vector::Add(Print\* p)

{

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

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

{

if (v.size == 0) { out << "" << endl; return out; }

Print\*\* p = v.beg;

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

{

(\*p)->print();

p++;

}

return out;

}

Triad.h

#pragma once

#include <iostream>

#include "Print.h"

using namespace std;

class Triad:public Print

{

protected:

int first;

int second;

int third;

public:

void print() override;

Triad();

Triad(int first, int second, int third);

Triad(const Triad&);

void setFirst(int f);

void setSecond(int s);

void setThird(int t);

void setTriad(int f, int s, int t);

friend std::istream& operator >> (std::istream& in, Triad&);

void plus();

~Triad() {};

};

Triad.cpp

#include <iostream>

#include "Triad.h"

#include "Print.h"

using namespace std;

Triad::Triad()

{

first = 0;

second = 0;

third = 0;

}

Triad::Triad(int f, int s, int t)

{

first = f;

second = s;

third = t;

}

Triad::Triad(const Triad& t)

{

first = t.first;

second = t.second;

third = t.third;

}

void Triad::print()

{

cout << "1) " << first << endl;

cout << "2) " << second << endl;

cout << "3) " << third << endl;

}

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

{

in >> t.first;

in >> t.second;

in >> t.third;

return in;

}

void Triad::plus()

{

first++;

second++;

third++;

}

void Triad::setFirst(int f)

{

first = f;

}

void Triad::setSecond(int s)

{

second = s;

}

void Triad::setThird(int t)

{

third = t;

}

void Triad::setTriad(int f, int s, int t)

{

first = f;

second = s;

third = t;

}

Data.h

#pragma once

#include <iostream>

#include "Triad.h"

using namespace std;

class Data : public Triad

{

int year;

int month;

int day;

public:

Data();

Data(int y, int m, int d);

Data(const Data&);

void plus();

void plusD(int d);

void print() override;

};

Data.cpp

#include <iostream>

#include "Data.h"

#include "Triad.h"

using namespace std;

Data::Data()

{

year = 0;

month = 0;

day = 0;

}

Data::Data(int y, int m, int d)

{

if (d > 30)

{

m = d / 30;

day = d % 30;

}

else

{

day = d;

}

if (m > 12)

{

y = m / 12;

month = m % 12;

}

else

{

month = m;

}

year = y;

}

Data::Data(const Data& d)

{

day = d.day;

month = d.month;

year = d.year;

}

void Data::print()

{

cout << year << ". " << month << ". " << day << endl;

}

void Data::plus()

{

year++;

month++;

day++;

}

void Data::plusD(int d)

{

if (day + d > 30)

{

month = (day + d) / 30;

day = (day + d) % 30;

if (month > 12)

{

year += month / 12;

month = month % 12;

}

}

else

{

day += d;

}

}

Main.cpp

#include <iostream>

#include "Triad.h"

#include "Data.h"

#include "Print.h"

#include "Vector.h"

using namespace std;

int main()

{

setlocale(0, "RUS");

Data d;

Triad t;

Vector v;

v.SetVectorSize(5);

v.SetVectorBeg();

Print\* p = &t;

v.Add(p);

p = &d;

v.Add(p);

Triad l;

p = &l;

v.Add(p);

cout << v;

}

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.