Министерство образования и науки Российской Федерации

Федеральное государственное образовательное автономное учреждение высшего образования

"Пермский национальный исследовательский политехнический университет"

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

Дисциплина: Основы алгоритмизации и программирования

Тема: Наследование. Виртуальные функции. Полиморфизм.

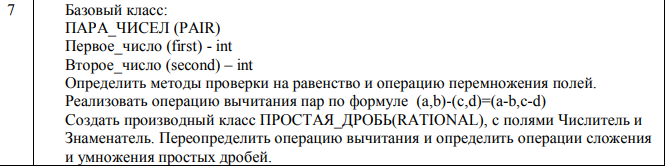
|  |  |
| --- | --- |
| Выполнил работу | |
| Студент группы РИС-22-1б | |
| Бреднев М.П. | |
|  | |
| Проверил работу | |
| Доцент кафедры ИТАС | |
| Полякова О.А. | |
|  | |

Пермь – 2023

**Анализ предметной области**

**Постановка задачи**

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 основнй функции продемонстрировать перегруженные операции и полиморфизм вектора.



**Анализ задачи**

Класс Pair представляет из себя класс человек, который может существовать отдельно, но имеет очень ограниченный функционал. Класс Pair – родитель, от которого можно унаследовать любое число и добавить к нему недостающие характеристики. Класс Rational – класс предок, который унаследован от класса Person и расширяет его функционал. У него добавляются 2 поля – chislitel и znamenatel, а также метод, который проверяет, есть ли знаменатель. Также определен абстрактный класс object, от которого наследуется остальные классы, исключая вектор. Указатели данного класса могут указывать на любой класс и вызывать методы класса, которые переопределены от виртуальных функций. На этом принципе будет основан класс Вектор.

**Тестирование программы**

****

*Рис. 1 – Тестирование программы.*

# Ответы на контрольные вопросы

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

Чисто виртуальный метод - виртуальная функция, не имеющая определения в базовом классе. Для страховки от неправильного вызова ее часто объявляют равной нулю: virtual <тип> <имя функции> (<список параметров>) = 0; Чисто виртуальный метод должен переопределяться в производном классе (возможно, опять как чисто виртуальный). Переопределение происходит с помощью спецификатора override (он не обязателен).

2. Какой класс называется абстрактным?

Абстрактный класс - класс, содержащий хотя бы один чисто виртуальный метод.

3. Для чего предназначены абстрактные классы?

Используется как обобщенная концепция, которая используется для создания конкретных производных классов. Создание объектов абстрактного класса невозможно, однако можно использовать указатели и типы на типы абстрактных классов. Абстрактные классы нельзя использовать для приведения типов, но допускается объявление ссылок и указателей, если абстрактный класс имеет наследников. Если производный класс не определяет все чисто виртуальные функции, он также является абстрактным

Пример:

Нужно сделать для класса TextFile базовый класс File, от которого будет унаследован еще один класс RTFFile. Однако, в такой ситуации неизвестно как реализовать метод read() класса File, т.к. класс File не реализует поведение какого-то конкретного типа файлов, а представляет интерфейс для работы с различными файлами. Для этого используется чисто виртуальная функция.

class File {

virtual string read(int count) = 0;

};

Метод read(...) должен быть определен в классах наследниках. Теперь класс File стал абстрактным, и его экземпляры невозможно создать. Но можно работать через указатель на абстрактный класс с объектами производных классов.

File \*f = new TextFile("text.txt");

//различные действия с файлом text.txt

delete f;

f = new RTFFile("rich\_text.rtf");

//различные действия с файлом rich\_text.rtf

delete f;

4. Что такое полиморфные функции?

При работе с абстрактными классами можно создать функцию, параметром которой будет являться указатель на абстрактный класс. Туда может передаваться указатель на объект любого производного класса, что позволяет создать полиморфные функции, то есть функции, работающие с объектом любого типа в пределах одной иерархии.

class Base {

virtual void work() = 0;

};

class Derived : Base {...};

void print(Base\*a){ cout << “Hello!”}

//в основной функции

Derived \*t = new Derived;

print(t); //хотя в функцию передается базовый файл в качестве аргумента.

5. Чем полиморфизм отличается от принципа подстановки?

Полиморфизм - это возможность классов иметь разную реализацию одного и того же функционала в основном с помощью наследования. Это механизм работы языка программирования.

Принцип подстановки является “правилом хорошего кода". В хорошем коде все методы класса-потомка должны быть применимы к объекту класса родителя.

6. Привести примеры иерархий с использованием абстрактных классов.

class Shape {

public:

virtual double getSquare() const = 0; // площадь фигуры

virtual double getPerimeter() const = 0; // периметр фигуры

};

class Rectangle : public Shape { // класс прямоугольника

public:

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

double getSquare() const override {

return width \* height;

}

double getPerimeter() const override {

return width \* 2 + height \* 2;

}

private:

double width; // ширина

double height; // высота

};

class Circle : public Shape { // круг

public:

Circle(double r) : radius(r) { }

double getSquare() const override {

return radius \* radius \* 3.14;

}

double getPerimeter() const override {

return 2 \* 3.14 \* radius;

}

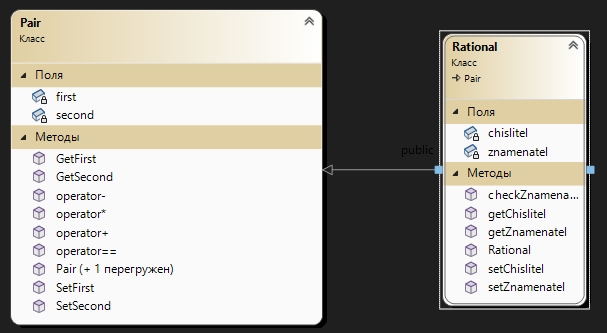
private:

double radius; // радиус круга

};

**Приложения**

Приложение UML-диаграмма



Приложение Б – код программы

Main.cpp:

#include <iostream>

#include "Header.h"

using namespace std;

class Rational : public Pair {

private:

int chislitel;

int znamenatel;

public:

Rational(int ch = 0, int zn = 0);

void setChislitel(int ch);

void setZnamenatel(int zn);

int getChislitel();

int getZnamenatel();

void checkZnamenatel();

friend istream& operator>>(istream& is, Rational& s);

};

Rational::Rational(int f, int s) : Pair(f, s) {

chislitel = f;

znamenatel = s;

}

void Rational::setChislitel(int ch) {

chislitel = ch;

}

void Rational::setZnamenatel(int zn) {

znamenatel = zn;

}

int Rational::getChislitel() {

return chislitel;

}

int Rational::getZnamenatel() {

return znamenatel;

}

void Rational::checkZnamenatel() {

if (znamenatel != 0) {

cout << "Знаменателя нет" << endl;

}

else {

cout << "Знаменатель есть" << endl;

}

}

istream& operator>>(istream& is, Rational& s) {

int ch, zn;

cout << "Введите числитель: ";

is >> ch;

s.SetFirst(ch);

cout << "Введите знаменатель: ";

is >> zn;

s.SetSecond(zn);

return is;

}

Rational operator- (const Rational& ch, const Rational& zn) {

return Rational(ch - zn);

}

Rational operator+ (const Rational& ch, const Rational& zn) {

return Rational(ch + zn);

}

Rational operator\* (const Rational& ch, const Rational& zn) {

return Rational(ch + zn);

}

int main() {

setlocale(0, "");

Pair a(1, 3);

Pair b(1, 3);

Pair c = a - b;

cout << c.GetFirst();

cout << " ";

cout << c.GetSecond();

cout << "\n\n";

Pair d = a \* b;

cout << d.GetFirst();

cout << " ";

cout << d.GetSecond();

cout << "\n\n";

if (a == b) cout << "YA";

else cout << "NU";

cout << "\n\n";

Rational chislo1(10, 5);

Rational chislo2(5, 5);

Rational chislo3(chislo1.getChislitel() - chislo2.getChislitel(), chislo1.getZnamenatel() - chislo2.getZnamenatel());

cout << chislo3.getChislitel() << " " << chislo3.getZnamenatel();

cout << "\n\n";

Rational chislo4(chislo1.getChislitel() + chislo2.getChislitel(), chislo1.getZnamenatel() + chislo2.getZnamenatel());

cout << chislo4.getChislitel() << " " << chislo4.getZnamenatel();

cout << "\n\n";

Rational chislo5(chislo1.getChislitel() \* chislo2.getChislitel(), chislo1.getZnamenatel() \* chislo2.getZnamenatel());

cout << chislo5.getChislitel() << " " << chislo5.getZnamenatel();

cout << "\n\n";

chislo1.checkZnamenatel();

return 0;

}

pair.h:

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Pair {

private:

int first;

int second;

public:

Pair()

{

}

Pair(int f, int s);

void SetFirst(int f);

void SetSecond(int s);

int GetFirst();

int GetSecond();

Pair operator+(const Pair& a);

Pair operator\*(const Pair&);

Pair operator-(const Pair&);

bool operator==(const Pair&);

};

pair.cpp:

#include "Header.h"

#include <iostream>

using namespace std;

Pair::Pair(int f, int s) {

first = f;

second = s;

}

void Pair::SetFirst(int f) {

first = f;

}

void Pair::SetSecond(int s) {

second = s;

}

int Pair::GetFirst() {

return first;

}

int Pair::GetSecond() {

return second;

}

Pair Pair::operator+(const Pair& a) {

Pair t;

t.first = first + a.first;

t.second = second + a.second;

return t;

}

Pair Pair :: operator\*(const Pair& a) {

Pair t;

t.first = first \* a.first;

t.second = second \* a.second;

return t;

}

Pair Pair :: operator-(const Pair& a) {

Pair t;

t.first = first - a.first;

t.second = second - a.second;

return t;

}

bool Pair :: operator==(const Pair& a)

{

if ((first == a.first) && (second == a.second)) return true;

else return false;

}