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 ПО ЛАБОРАТОРНОЙ РАБОТЕ №5**

Дисциплина: «Основы алгоритмизации и программирования»

Тема : "Наследование. Виртуальные функции. Полиморфизм"

Выполнил работу:

Студент группы РИС-22-1Б

Батин В.В.

Проверил:

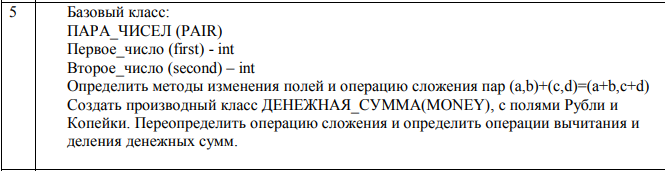
Доцент кафедры ИТАС

Полякова О.А.

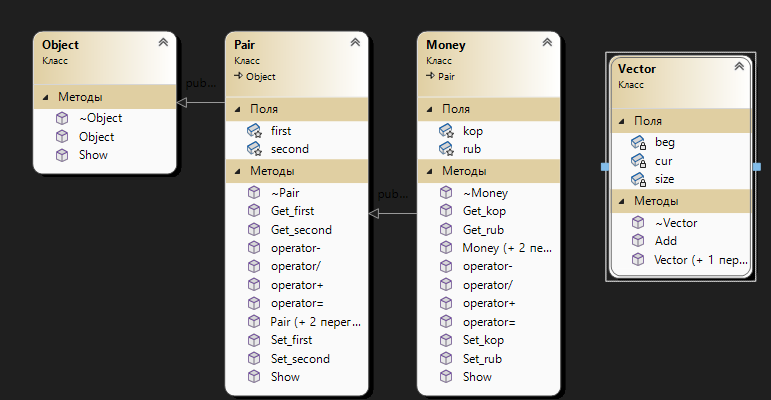
Г. Пермь-2023

**Постановка задачи**

1. Определить абстрактный класс.  
2. Определить иерархию классов, в основе которой будет находиться абстрактный класс (см. лабораторную работу No4).  
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.



**Диаграмма классов**

****

**Код**

**Описание класса Pair**

#pragma once

#include "Object.h"

#include <iostream>

using namespace std;

class Pair :public Object

{

public:

Pair() {first = 0; second = 0;}//к

~Pair() {};

void Show() override;//функция для просмотра атрибутов класса с помощью указателя

Pair operator+(const Pair&);

Pair operator-(const Pair&);

Pair operator/(const Pair&);

Pair(int F, int S) {first = F; second = S;}//к с параметрами

Pair(const Pair& Pair) {first = Pair.first; second = Pair.second;}//копирование

int Get\_first() { return first; }

int Get\_second() { return second; }

void Set\_first(int F) { first = F; }

void Set\_second(int S) { second = S; }

Pair& operator=(const Pair&);

friend istream& operator>>(istream& in, Pair& c);

friend ostream& operator<<(ostream& out, const Pair& c);

protected:

int first;

int second;

};

**Определение методов класса Pair**

#include "Pair.h"

Pair& Pair::operator=(const Pair& c)

{

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

first = c.first;

second = c.second;

return \*this;

}

Pair Pair::operator+(const Pair& t)

{

Pair p;

p.first = first + t.first;

p.second = second + t.second;

return p;

}

Pair Pair::operator-(const Pair& t)

{

Pair p;

p.first = first - t.first;

p.second = second - t.second;

return p;

}

Pair Pair::operator/(const Pair& t)

{

Pair p;

p.first = first / t.first;

p.second = second / t.second;

return p;

}

istream& operator>>(istream& in, Pair& c)

{

cout << "\nfirst:"; in >> c.first;

cout << "\nsecond:"; in >> c.second;

return in;

}

ostream& operator<<(ostream& out, const Pair& c)

{

out << "\nfirst : " << c.first;

out << "\nsecond : " << c.second;

out << "\n";

return out;

}

void Pair::Show()

{

cout << "\nfirst : " << first;

cout << "\nsecond : " << second;

cout << "\n";

}

**Описание класса Money**

#pragma once

#include "Pair.h"

class Money : public Pair

{

public:

Money() { rub = 0; kop = 0; }

~Money() { };

void Show() override;//функция для просмотра атрибутов класса с помощью указателя

Money(int F, int S) : Pair(F,S) { rub = F; kop = S; }

Money(const Money& Money) { rub = Money.rub; kop = Money.kop; }//копирование

int Get\_rub() { return rub; }

int Get\_kop() { return kop; }

void Set\_rub(int R) { rub = R; }

void Set\_kop(int K) { kop = K; }

Money& operator=(const Money&);

Money operator+(const Money&);

Money operator-(const Money&);

Money operator/(const Money&);

friend istream& operator>>(istream& in, Money& m);

friend ostream& operator<<(ostream& out, const Money& m);

protected:

int rub;

int kop;

};

**Определение методов класса Money**

#include "Money.h"

Money& Money::operator=(const Money& c)

{

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

rub = c.rub;

kop = c.kop;

return \*this;

}

istream& operator>>(istream& in, Money& c)

{

cout << "\nrub:"; in >> c.rub;

cout << "\nkop:"; in >> c.kop;

return in;

}

ostream& operator<<(ostream& out, const Money& c)

{

out << "\nrub : " << c.rub;

out << "\nkop : " << c.kop;

out << "\n";

return out;

}

Money Money::operator+(const Money& t)

{

Money p;

p.rub = rub + t.rub;

p.kop = kop + t.kop;

return p;

}

Money Money::operator-(const Money& t)

{

Money p;

p.rub = rub - t.rub;

p.kop = kop - t.kop;

return p;

}

Money Money::operator/(const Money& t)

{

Money p;

p.rub = rub / t.rub;

p.kop = kop / t.kop;

return p;

}

void Money::Show()

{

cout << "\nrub : " << rub;

cout << "\nkop : " << kop;

cout << "\n";

}

**Описание класса Object**

#pragma once

class Object

{

public:

Object()

{

}

~Object()

{

}

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

};

**Описание класса Object**

#pragma once

#include "Object.h"

#include <string>

#include <iostream>

using namespace std;

class Vector

{

public:

Vector();//конструктор без параметров

Vector(int);//конструктор копирования

public:

~Vector();//деструктор

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

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

private:

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

int size;//размер

int cur;//текущая позиция

};

**Определение методов класса Vector**

#include "Vector.h"

//конструктор без параметров

Vector::Vector(void)

{

beg = 0;

size = 0;

cur = 0;

}

//деструктор

Vector::~Vector(void)

{

if (beg != 0)delete[] beg;

beg = 0;

}

//конструктор с параметрами

Vector::Vector(int n)

{

beg = new Object \* [n];

cur = 0;

size = n;

}

//добавление объекта, на который указывает указатель p в вектор

void Vector::Add(Object\* p)

{

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

//операция вывода

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

{

if (v.size == 0) out << "Empty" << endl;

Object\*\* p = v.beg;//указатель на указатель типа Object

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

{

(\*p)->Show();//вызов метода Show() (позднее связывание)

p++;//передвигаем указатель на следующий объект

}

return out;

}

**int main()**

//Базовый класс :

//ПАРА\_ЧИСЕЛ(PAIR)

//Первое\_число(first) - int

//Второе\_число(second) – int

//Определить методы изменения полей и операцию сложения пар(a, b) + (c, d) = (a + b, c + d)

//Создать производный класс ДЕНЕЖНАЯ\_СУММА(MONEY), с полями Рубли и

//Копейки.Переопределить операцию сложения и определить операции вычитания и

//деления денежных сумм.

#include "Object.h"

#include "Pair.h"

#include "Money.h"

#include "Vector.h"

#include <iostream>

using namespace std;

void main()

{

Vector(3);

Vector v(5);//вектор из 5 элементов

Pair a;//объект класса Car

cin >> a;

Money b;// объект класса Lorry

cin >> b;

Object\* p = &a;//ставим указатель на объект класса Pair

v.Add(p);//добавляем объект в вектор

p = &b;//ставим указатель на объект класса Money

v.Add(p); //добавляем объект в вектор

cout << v;//вывод вектора

Pair t1;

cin >> t1;

cout << t1;

Pair t2(4, 6);

cout << endl;

cout << t1 + t2 << endl;

cout << t1 - t2 << endl;

cout << t1 / t2 << endl;

Money t3;

cin >> t3;

cout << t3;

cout << endl;

Money t4(2, 10);

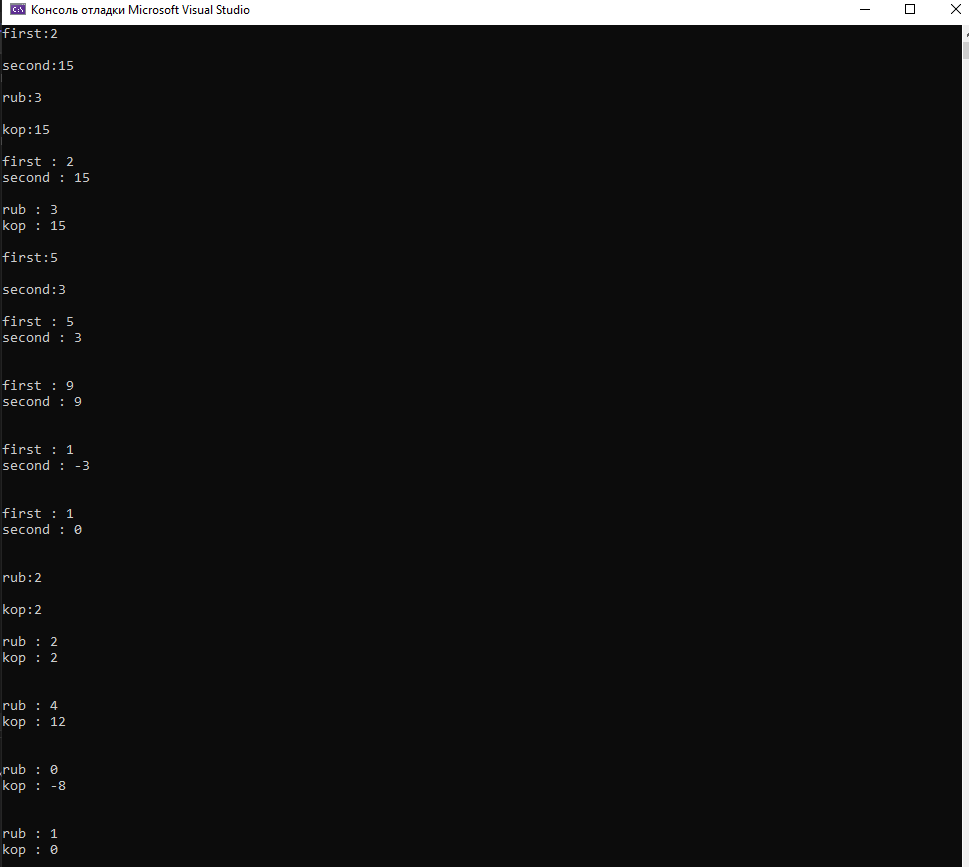
cout << t3 + t4 << endl;

cout << t3 - t4 << endl;

cout << t3 / t4 << endl;

}

**Результат выполнения**



**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.

 Полиморцная 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 ф-ия 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()

    {

        std::cout << "Rectangle" << std::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()

    {

        std::cout << "Circle" << std::endl;

    }

};

int main()

{

    Rectangle rect(30, 50);

    Circle circle(30);

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

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