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: 09.03.01 - «Информатика и вычислительная техника»

**Отчет по лабораторной работе “Классы” №5**

**по дисциплине**

**«Теория алгоритмов и структуры данных»**

Выполнил студент гр. ИВТ-21-1б

Ахунов Руслан Булатович

Проверил:

Ст. Преподаватель кафедры ИТАС

Яруллин Денис Владимирович

(оценка) (подпись)

г. Пермь - 2022

**Постановка задачи:**

1.  Определить абстрактный класс.

2.  Определить иерархию классов, в основе которой будет находиться абстрактный класс (см. лабораторную работу №4).

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.

Вариант 2:

Базовый класс:

ПАРА\_ЧИСЕЛ (PAIR)

Первое\_число (first) - int Второе\_число (second) – int

Определить методы изменения полей и вычисления произведения чисел. Создать производный класс ПРЯМОУГОЛЬНИК (RECTANGLE), с полями- сторонами. Определить методы для вычисления площади и периметра прямоугольника.

**Анализ задачи:**

class abstract

{

public:

    virtual void show() = 0;

};

Класс abstract, как понятно из названия является абстрактным классом, в котором есть только один метод show.

Следующие 2 класса является копией из предыдущей лабораторной работы, единственные изменения это добавления метода show

void show() override

    {

        cout << first << " , " << second << endl;

    }

class Vector

{

private:

    abstract\*\* beg;

    int size;

    int cur;

public:

    Vector()

    {

        beg = 0;

        size = 0;

        cur = 0;

    }

    Vector(int n)

    {

        beg = new abstract \* [n];

        cur = 0;

        size = n;

    }

    void add(abstract\* tmp)

    {

        if (cur < size)

        {

            beg[cur] = tmp;

            cur++;

        }

    }

    ~Vector()

    {

        if (beg != 0)

        {

            delete[] beg;

        }

        beg = 0;

    }

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

};

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

{

    cout << "Вектор:" << endl;

    if (v.size == 0)

    {

        out << "Empty" << endl;

    }

    abstract\*\* tmp = v.beg;

    for (int i = 0; i < v.cur; i++)

    {

        (\*tmp)->show();

        tmp++;

    }

    return out;

}

Класс Vector, котором есть поля beg – массив указателей на элементы, size – размер вектора, cur – текущий порядковый номер элемента. Так же был перегружен оператор потокового вывода, где был использован метод show().

int main()

{

    setlocale(LC\_ALL, "Rus");

    Vector v(5);

    Pair a;

    cin >> a;

    Rect b;

    cin >> b;

    abstract\* tmp = &a;

    v.add(tmp);

    tmp = &b;

    v.add(tmp);

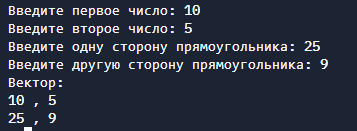
    cout << v;

    return 0;

}

В функции main, мы инициализируем переменную v типа Vector, с помощью конструктора с параметром. Затем объявляем и инициализируем две переменные a и b, типа Pair и Rect соответственно. А также временную переменную tmp типа abstract, при этом эта переменная является указателем.

**Результат работы программы**

****

Как можно видеть, при выводе вектора, вступают в силу тот метод show, который есть в данном классе.

**Код программы**

#include <iostream>

using namespace std;

class abstract

{

public:

    virtual void show() = 0;

};

class Pair : public abstract

{

protected:

    int first;

    int second;

public:

    void show() override

    {

        cout << first << " , " << second << endl;

    }

    Pair()

    {

        first = 0;

        second = 0;

    }

    Pair(int f, int s)

    {

        first = f;

        second = s;

    }

    Pair(const Pair& tmp)

    {

        first = tmp.first;

        second = tmp.second;

    }

    void setFirst(int f)

    {

        first = f;

    }

    void setSecond(int s)

    {

        second = s;

    }

    int getFirst() { return first; }

    int getSecond() { return second; }

    Pair& operator=(const Pair& p)

    {

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

        first = p.first;

        second = p.second;

        return \*this;

    }

    Pair operator\*(const Pair tmp)

    {

        first = first \* tmp.first;

        second = second \* tmp.second;

        return \*this;

    }

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

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

    virtual ~Pair()

    {

        // cout << "Удалили Pair..." << endl << endl;

    }

};

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

{

    cout << "Введите первое число: "; in >> tmp.first;

    cout << "Введите второе число: "; in >> tmp.second;

    return in;

}

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

{

    return (out << "(" << tmp.first << "," << tmp.second << ")");

}

class Rect : public Pair

{

public:

    void show() override

    {

        cout << first << " , " << second << endl;

    }

    int P()

    {

        int p;

        p = (first + second) \* 2;

        return p;

    }

    int S()

    {

        int s;

        s = first \* second;

        return s;

    }

    Rect()

    {

        first = 0;

        second = 0;

    }

    Rect(int f, int s)

    {

        first = f;

        second = s;

    }

    Rect(const Rect& tmp)

    {

        first = tmp.first;

        second = tmp.second;

    }

    void setFirst(int f)

    {

        first = f;

    }

    void setSecond(int s)

    {

        second = s;

    }

    int getFirst() { return first; }

    int getSecond() { return second; }

    Rect& operator=(const Rect& p)

    {

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

        first = p.first;

        second = p.second;

        return \*this;

    }

    friend istream& operator>>(istream& in, Rect& tmp);

    friend ostream& operator<<(ostream& out, const Rect& tmp);

    ~Rect()

    {

        // cout << "Удалили Rect..." << endl << endl;

    }

};

istream& operator>>(istream& in, Rect& tmp)

{

    cout << "Введите одну сторону прямоугольника: "; in >> tmp.first;

    cout << "Введите другую сторону прямоугольника: "; in >> tmp.second;

    return in;

}

ostream& operator<<(ostream& out, const Rect& tmp)

{

    return (out << "Одна сторона равна: " << tmp.first << " / " << "Другая стороная равна: " << tmp.second);

}

class Vector

{

private:

    abstract\*\* beg;

    int size;

    int cur;

public:

    Vector()

    {

        beg = 0;

        size = 0;

        cur = 0;

    }

    Vector(int n)

    {

        beg = new abstract \* [n];

        cur = 0;

        size = n;

    }

    void add(abstract\* tmp)

    {

        if (cur < size)

        {

            beg[cur] = tmp;

            cur++;

        }

    }

    ~Vector()

    {

        if (beg != 0)

        {

            delete[] beg;

        }

        beg = 0;

    }

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

};

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

{

    cout << "Вектор:" << endl;

    if (v.size == 0)

    {

        out << "Empty" << endl;

    }

    abstract\*\* tmp = v.beg;

    for (int i = 0; i < v.cur; i++)

    {

        (\*tmp)->show();

        tmp++;

    }

    return out;

}

int main()

{

    setlocale(LC\_ALL, "Rus");

    Vector v(5);

    Pair a;

    cin >> a;

    Rect b;

    cin >> b;

    abstract\* tmp = &a;

    v.add(tmp);

    tmp = &b;

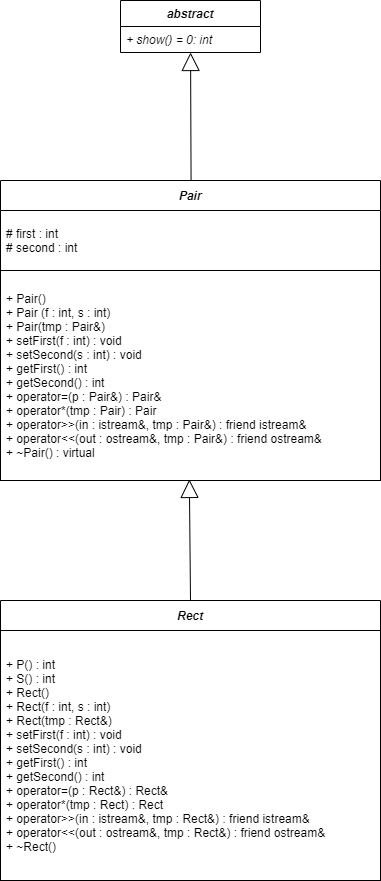
    v.add(tmp);

    cout << v;

    return 0;

}

**Диаграмма класса**



**Вопросы**

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

Чисто виртуальный метод содержит признак = 0 вместо тела, например:

virtual void f(int) = 0;

Чисто виртуальный метод должен переопределяться в производном классе (возможно, опять как чисто виртуальный).

* 1. **Какой класс называется абстрактным?**

Класс, который имеет хотя бы один чисто виртуальный метод.

* 1. **Для чего предназначены абстрактные классы?**

Абстрактные классы предназначены для представления общих понятий, которые предполагается конкретизировать в производных классах.

* 1. **Что такое полиморфные функции?**

Функции, которые способны работать с объектом любого типа в пределах одной иерархии

* 1. **Чем полиморфизм отличается от принципа подстановки?**

Принцип подстановки отвечает за спецификацию классов, а полиморфизм за их поведение.

* 1. **Привести примеры иерархий с использованием абстрактных классов.**

В абстрактном классе есть метод «use», далее создаем несколько наследуемых классов оружия, например нож и пистолет, мы можем использовать метод use при этом перезаписав его в соответствующих классов, то есть нож – ударяет, пистолет – стреляет.