Частное учреждение образования
«Колледж бизнеса и права»

УТВЕРЖДАЮ
Заведующий методическим кабинетом
_____ Е.В. Фалей
«_____»_____ 2017 г.

Специальность: 2-40 01 01 «Программное обеспечение информационных технологий»	Дисциплина: «Основы алгоритмизации и программирование»

Лабораторная работа № 26
Инструкционно-технологическая карта

Тема: Разработка алгоритмов и программ с использованием чистых виртуальных методов

Цель: Научиться создавать программы и классы с использованием чистых виртуальных методов

Время выполнения: 2 часа

1. Краткие теоретические сведения

Механизм **виртуальных методов** (ВМ) в ООП используется для реализации полиморфизма: создания метода, предназначенного для работы с различными объектами из иерархии наследования за счет механизма позднего (динамического) связывания. Виртуальные методы объявляются в базовом (родительском) и производном (дочернем) классах с ключевым словом **virtual** перед определением ВМ, имеют одинаковое имя, список аргументов (сигнатуру) и возвращаемое значение. Метод, объявленный в базовом классе как виртуальный, будет виртуальным во всех производных классах, где он встретится, даже если слово **virtual** в дочерних классах будет отсутствовать.

В отличие от механизма перегрузки функций (функции с одним и тем же именем, но с различными типами входных аргументов считаются разными) виртуальные методы объявляются в порожденных (дочерних) классах с тем же именем, возвращаемым значением и типом аргументов (сигнатурой). Если различны типы входных аргументов, виртуальный механизм игнорируется. Тип возвращаемого значения переопределить нельзя.

Класс, содержащий хотя бы один **чисто виртуальный метод** (ЧВМ), называется ***абстрактным***. ***Чисто виртуальный метод*** всегда содержит признак, по которому его можно определить как ЧВМ – код ”**= 0;**” вместо тела {…}, например:

virtual void MyMethod(int) = 0;

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

Абстрактные классы предназначены для представления общих понятий, которые предполагается конкретизировать в производных классах. Абстрактный класс может использоваться ***только в качестве базового (родительского)*** для других классов – объекты абстрактного класса создавать **нельзя**, поскольку прямой или косвенный

вызов чисто виртуального метода приводит к ошибке при выполнении программы с таким кодом.

При определении абстрактного класса необходимо иметь в виду следующее:

- абстрактный класс нельзя использовать при явном приведении типов, для описания типа параметра и типа возвращаемого методом значения;
- допускается объявлять указатели и ссылки типа абстрактного класса, если при их инициализации не требуется создавать временный объект типа абстрактного класса;
- если класс, производный от абстрактного, не определяет с телами **все** унаследованные им чисто виртуальные методы, то он также является абстрактным.

Таким образом, можно создать функцию или метод, параметром которого является указатель на абстрактный класс. На место этого параметра при выполнении программы может передаваться указатель на объект любого производного класса из данной иерархии наследования. Это позволяет создавать ***полиморфные функции***, работающие с объектом любого типа в пределах одной иерархии наследования.

2. Пример выполнения программы

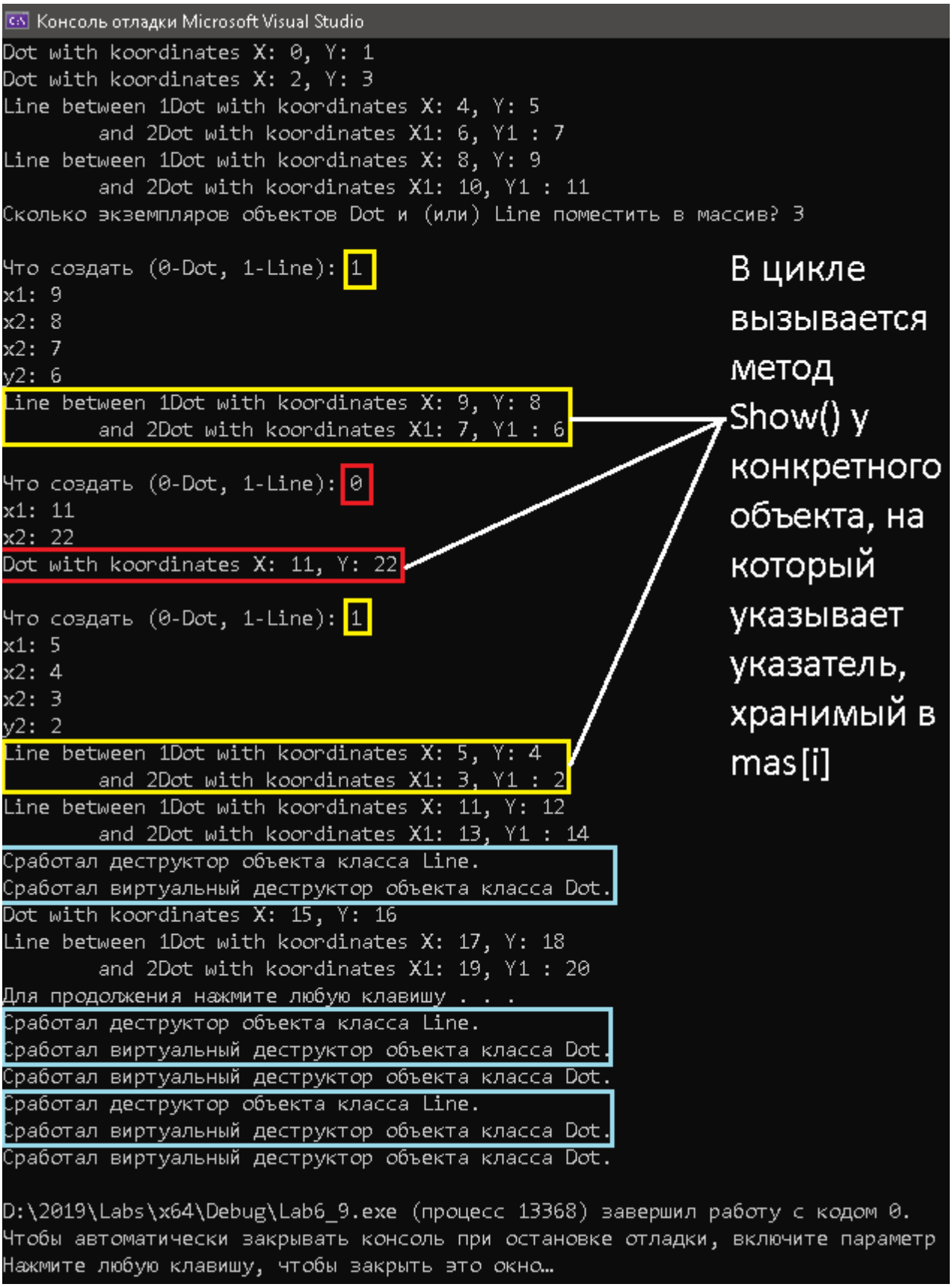
Задание 1.

Наберите код нижеследующих программ, *если не набрали его в лабораторной работе № 25*:

```cpp
#include <iostream>
#include <Windows.h>
using namespace std;

class Dot
{
protected:
    int x, y;
public:
    virtual void Show()//этот метод можно вызвать у экземпляра данного класса, но еще он указан как виртуальный,
    {//т.е. он может быть переопределен в дочерних классах и пригоден для позднего связывания
        cout << "Dot with koordinates X: " << x << ", Y: " << y << endl;
    }
    Dot(int x0, int y0) : x(x0), y(y0)//параметризованный конструктор с прямой инициализацией
    {//его тело пустое, поскольку код в строке выше все 2 поля проинициализировал
    }
    Dot()//нужен конструктор без параметров, чтобы создать массив из указателей класса Точка
    {//поскольку выше мы написали конструктор с параметрами, код которого "скрыл" автоматический конструктор без параметров,
    }//то надо самим написать конструктор без параметров (тело у него пустое)
    //~Dot()//невиртуальный деструктор класса Dot. Поскольку класс может иметь только один деструктор,то один из двух деструкторов должен быть закомментирован(скрыт от компилятора)
    //{
    //  cout << "Сработал деструктор объекта класса Dot.\n";
    //}
    virtual ~Dot()//виртуальный деструктор класса Dot. Конструкторы НЕ могут быть виртуальными, а деструкторы могут и должны быть виртуальными, если содержащие их классы имеют
    {//виртуальные методы
        cout << "Сработал виртуальный деструктор объекта класса Dot.\n";
    }
};

class Line : public Dot//наследование открытое (public), а если сделать его закрытым (private) или защищенным (protected),
{//то метод Show() получит соответствующий спецификатор и станет недоступным из функции main()
protected:
    int x1, y1;//поля x и y в этом классе есть благодаря наследованию, нужно только досоздать недостающие нужные нам поля
public:
    void Show() override//override указывает компилятору проверить, действительно ли в родительском классе есть виртуальный метод
    {//с таким же именем и сигнатурой (списком входных параметров), и после этого переопределить (заместить) его тут новой реализацией
        cout << "Line between 1";
```

```cpp
        Dot::Show();//вызываем метод родительского класса, используя его код повторно
        cout << "\tand 2Dot with koordinates X1: " << x1 << ", Y1 : " << y1 << endl;//и дописываем недостающий код для ДАННОГО класса
    }
    Line(int x0, int y0, int x2, int y2) :Dot(x0, y0), x1(x2), y1(y2)//делегирующий конструктор, вызывающий родительский конструктор Dot() для инициализации двух полей x и y,
    {//унаследованных от родительского класса Dot; также конструктор Line() является параметризованным конструктором, поскольку, после того, как отработал конструктор Dot(),
    }//конструктору Line() надо проинициализировать входными значениями еще два поля x1 и y1, но он это делает не в своем теле {}, а путем вызова кода: x1(x2), y1(y2),
    //эквивалентного коду: x1 = x2; y1 = y2;, в результате чего по итогу заполняются значениями все 4 поля объекта класса Line и тело самого конструктора Line() может оставаться
    //пустым, поскольку работа по инициализации всех полей создаваемого объекта уже выполнилась в 41-й строке кода
    ~Line()//деструктор класса Line. Можно переопределить деструктор класса Dot
    {
        cout << "Сработал деструктор объекта класса Line.\n";
    }
};

int main()
{
    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);
    Dot a(0, 1);//создать "статический" объект типа класса Dot (ТОЧКА) с именем a путем вызова конструктора Dot(int, int) с двумя параметрами
    Dot* b = new Dot(2, 3);//создать "динамический" объект типа класса Dot с именем b, для которого вызывается конструктор Dot(int, int) с двумя параметрами
    if (b == nullptr)//аналогично if (b == NULL)//проверка на успешность выделения динамической памяти под объект
    {
        cout << "Экземпляр класса Dot создать не удалось.\n";//сообщаем пользователю о ситуации
        system("pause");
        return 0;//завершаем программу
    }
    Line c(4, 5, 6, 7);//создать "статический" объект типа класса Line(ЛИНИЯ) с именем c путем вызова конструктора Line(int, int, int, int) с четырьмя параметрами для инициализации
    //всех четырех полей объекта класса ЛИНИЯ
    Line* d = new Line(8, 9, 10, 11);//создать "динамический" объект типа класса Line(ЛИНИЯ) с именем d путем вызова конструктора Line(int, int, int, int) с четырьмя параметрами
    if (d == nullptr)//аналогично if (d == NULL)//проверка на успешность выделения динамической памяти под объект
    {
        cout << "Экземпляр класса Line создать не удалось.\n";//сообщаем пользователю о ситуации
        system("pause");
        return 0;//завершаем программу
    }
    a.Show();//вызываем метод у "статического" объекта a
    b->Show();//вызываем метод у "динамического" объекта b
    c.Show();//вызываем метод у "статического" объекта c
    d->Show();//вызываем метод у "динамического" объекта d
    int n, f, xA, xB, yA, yB;
    do
    {
        cout << "Сколько экземпляров объектов Dot и (или) Line поместить в массив? ";
        cin >> n;
    } while (n < 1);
    Dot** mas = new Dot * [n];//создаем одномерный динамический массив из указателей типа Dot, которые могут благодаря наследованию и полиморфизму указывать на любой объект из
    //иерархии наследования Dot - Line
    for (int i = 0; i < n; i++)
    {
        cout << "\nЧто создать (0-Dot, 1-Line): ";//пользователь выбирает, что создать: Точку (0) или Линию (1)
        cin >> f;//сохраняем значение, определяющее выбор пользователя, в переменную f
        cout << "x1: ";//и у Точки и у Линии есть хотя бы одна точка (x, y), поэтому запрашиваем ее значения у пользователя в любом случае
        cin >> xA;
        cout << "x2: ";
        cin >> yA;
        if (f == 0)//if(!f)
        {
            mas[i] = new Dot(xA, yA);//создаем объект типа класса Точка и его адрес присваиваем очередному элементу массива (в указатель типа Dot*)
            if (mas[i] == nullptr)
            {
                cout << "Экземпляр класса Dot создать не удалось.\n";//сообщаем пользователю о ситуации
                system("pause");
                return 0;//завершаем программу
            }
        }
        else
        {
            cout << "x2: ";//для Линии надо еще значения для второй точки, запрашиваем их у пользователя
            cin >> xB;
            cout << "y2: ";
            cin >> yB;
            mas[i] = new Line(xA, yA, xB, yB);//создаем объект типа класса Линия и его адрес присваиваем очередному элементу массива (в указатель типа Dot*). Класс Line наследует
            //классу Dot, а потому объекты дочернего класса Линия можно присваивать указателям типа родительского класса Точка
            if (mas[i] == nullptr)
            {
                cout << "Экземпляр класса Line создать не удалось.\n";//сообщаем пользователю о ситуации
                system("pause");
                return 0;//завершаем программу
            }
        }
        mas[i]->Show();//метод Show() вызывается у очередного элемента массива mas[i], который представляет из себя указатель типа Dot*, указывающий или на объект типа Dot, или на
        //объект типа Line. Благодаря полиморфизму виртуального метода Show(), который имеет для каждого из этих родственных классов конкретную реализацию (изначальную для класса
        //Dot или переопределенную для класса Line) и благодаря механизму ПОЗДНЕГО СВЯЗЫВАНИЯ только в момент выполнения метода Show() определяется, у объекта какого именно класса
        //этот метод сейчас вызывается в момент исполнения программы и, соответственно, если очередной объект оказывается Точкой, то у него вызывается метод Show() из класса Точка,
        //а если очередной объект окажется Линией, то у него вызовется метод Show() из класса Линия, но поскольку это виртуальный метод с одинаковым именем и сигнатурой, то в коде
        //мы смогли не писать никаких проверок реального типа объекта, а воспользовались краткой записью mas[i]->Show(); при которой полиморфный метод будет работать для того
        //класса, к которому будет принадлежать конкретный экземпляр класса в момент выполнения программы в будущем, а не сейчас при наборе кода или компиляции .exe-файла.
        //до запуска программы пользователем неизвестно, объекты с какими индексами станут Точками, а какие - Линиями. Может, они все будут Линиями по решению пользователя

    }
    Dot* ptr;//создадим указатель типа родительского класса Dot
    ptr = new Line(11, 12, 13, 14);//присвоим указателю ptr адрес динамического объекта типа класса Line
    if (ptr == nullptr)//сделаем проверку на выделение оперативной памяти
    {
        cout << "Экземпляр класса Line создать не удалось.\n";//сообщаем пользователю о ситуации
        system("pause");
        return 0;//завершаем программу
    }
    ptr->Show();//вызовем метод Show() у объекта, на который указывает указатель ptr. Этот объект типа класса Линия!
    delete ptr;//освобождаем память от объекта, на который указывает указатель ptr. Сначала сработает деструктор класса Line и вслед за ним сработает деструктор класса Dot
    Dot e(15, 16);//создадим объект типа класса Dot
    ptr = &e;//присвоим указателю ptr адрес объекта e (это Точка)
    ptr->Show();//через указатель ptr вызовем у точки e метод Show()
    Line g(17, 18, 19, 20);//создадим объект g типа класса Line
    ptr = &g;//присвоим указателю ptr адрес объекта g (это Линия)
    ptr->Show();//через указатель ptr вызовем у линии g метод Show(). В каждом случае должен срабатывать метод Show() ТОГО КЛАССА, у объекта которого он фактически вызывается
    //благодаря полиморфному поведению и позднему связыванию
    system("pause");
    return 0;
}
```

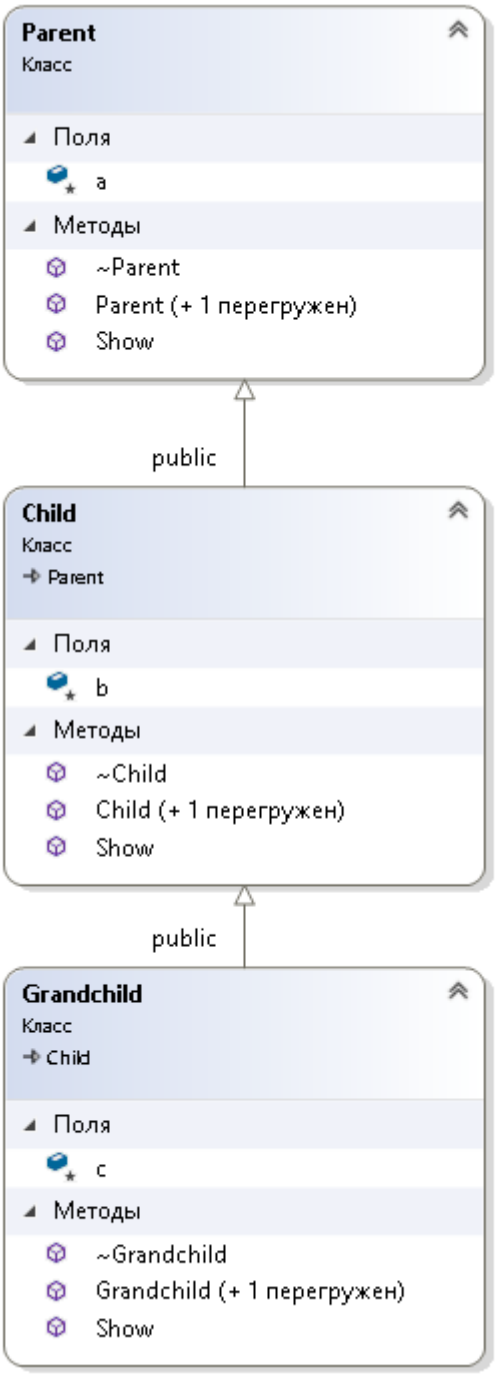
Тестируем программу:



class Base//образец кода (изучить; можно не набирать, поскольку члены классов //тут описаны, но не определены с телами)
{
public:
Base(); // конструктор без параметров по умолчанию

```
        Base(const Base&); // конструктор копирования
        virtual ~Base(); // виртуальный деструктор
        virtual void Show()=0; // чистый виртуальный метод
        // другие чистые виртуальные методы
protected:
        // защищенные члены класса
private:
        // часто остается пустым, иначе будет мешать будущим разработкам
};
class Derived: virtual public Base
{
public:
        Derived(); // конструктор без параметров по умолчанию
        Derived(const Derived&); // конструктор копирования
        Derived(параметры); // конструктор с параметрами
        virtual ~Derived(); // виртуальный деструктор
        void Show(); // переопределенный виртуальный метод
        // другие переопределенные виртуальные методы
        // другие перегруженные операторы
protected:
        // используется вместо private, если ожидается наследование
private:
        // используется для закрытых внутренних деталей реализации класса
};
```

Пример кода и диаграммы классов программы с абстрактным классом и унаследованными от него «обычными» классами (изучите и наберите код ниже):

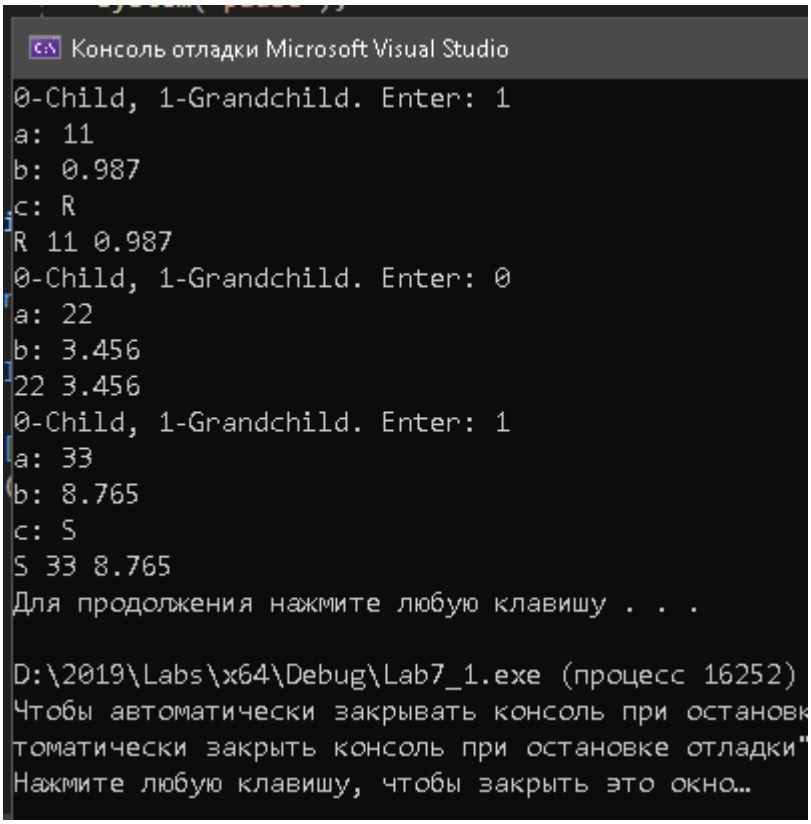

Parent
Класс
Поля
a
Методы
~Parent
Parent (+ 1 перегружен)
Show
public
Child
Класс
Parent
Поля
b
Методы
~Child
Child (+ 1 перегружен)
Show
public
Grandchild
Класс
Child
Поля
c
Методы
~Grandchild
Grandchild (+ 1 перегружен)
Show

```cpp
#include <iostream>
#include <Windows.h>
using namespace std;

class Parent
{
protected:
    int a = 0;
public:
    virtual void Show() = 0;
    Parent()
    {}
    Parent(int a0)
    {
        a = a0;
    }
    virtual ~Parent()
    {}
};
class Child :public Parent
{
protected:
    double b = 0;
public:
    void Show()override
    {
        cout << Parent::a << ' ' << Child::b << endl;
    }
    Child():Parent()
    {}
    Child(int a1, double b2) :Parent(a1)
    {
        b = b2;
    }
    virtual ~Child()
    {}
};
```

```
38  class Grandchild :public Child
39  {
40  protected:
41      char c = 'X';
42  public:
43      void Show()override
44      {
45          cout << Grandchild::c << ' ';
46          Child::Show();
47      }
48      Grandchild():Child()
49      {}
50      Grandchild(int a3, double b3, char c3) :Child(a3, b3)
51      {
52          c = c3;
53      }
54      virtual ~Grandchild()
55      {}
56  };
57  int main()
58  {
59      SetConsoleOutputCP(1251);
60      SetConsoleCP(1251);
61      //Parent a;//error
62      Child a;
63      Grandchild b;
64      int n = 3, p, a4;
65      double b4;
66      char c4;
67      Parent** m = new Parent*[n];
68      if (m == nullptr)
69      {
70          cout << "No memory.\n";
71          system("pause");
72          return 0;
73      }
74      for (int i = 0; i < n; i++)
```

```
75      {
76          cout << "0-Child, 1-Grandchild. Enter: ";
77          cin >> p;
78          switch (p)
79          {
80          case 0:
81          {
82              cout << "a: ";
83              cin >> a4;
84              cout << "b: ";
85              cin >> b4;
86              m[i] = new Child(a4, b4);
87              if (m[i] == nullptr)
88              {
89                  cout << "No memory.\n";
90                  system("pause");
91                  return 0;
92              }
93              break;
94          }
95          case 1:
96          {
97              cout << "a: ";
98              cin >> a4;
99              cout << "b: ";
100             cin >> b4;
101             cout << "c: ";
102             cin >> c4;
103             m[i] = new Grandchild(a4, b4, c4);
104             if (m[i] == nullptr)
105             {
106                 cout << "No memory.\n";
107                 system("pause");
108                 return 0;
109             }
110             break;
111         }
112         }
113         m[i]->Show();
114     }
115     for (int i = 0; i < n; i++)
116     {
117         delete[] m[i];
118     }
119     delete[] m;
120     system("pause");
121     return 0;
122 }
```

Тестируем:

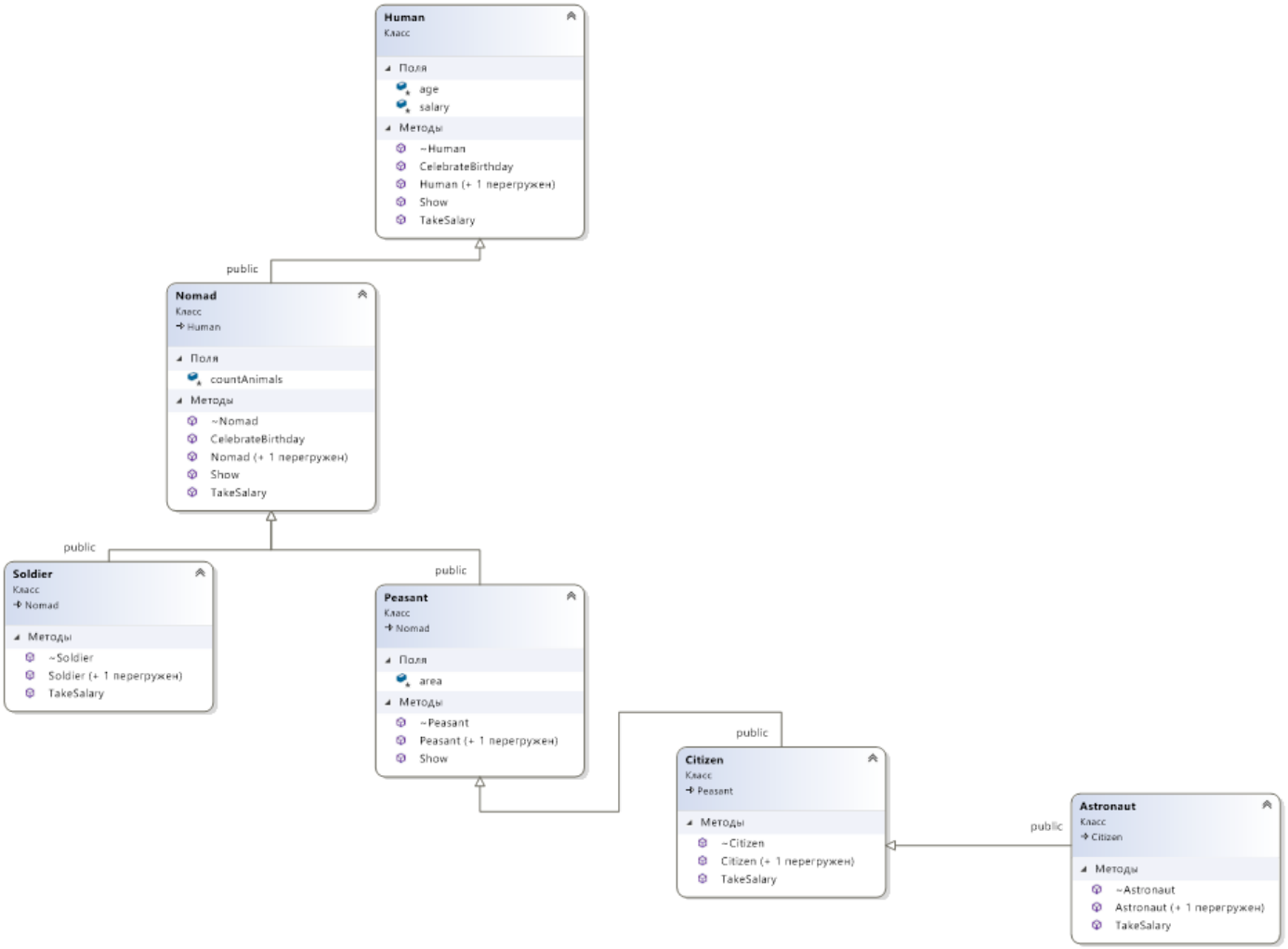


Задание 2.

Наберите код нижеследующей программы и на ее основе разберите абстрактные классы, чистые виртуальные методы, полиморфные функции и массивы, работу конструкторов и деструкторов классов в иерархии наследования. Протестируйте работу этой программы на своем наборе данных.

Ниже приведена диаграмма классов (иерархия классов в соответствии с их отношениями наследования), созданных в программе далее.

Далее пример кода программы, использующей абстрактный класс, чистые виртуальные методы и виртуальные переопределенные методы, полиморфную функцию:



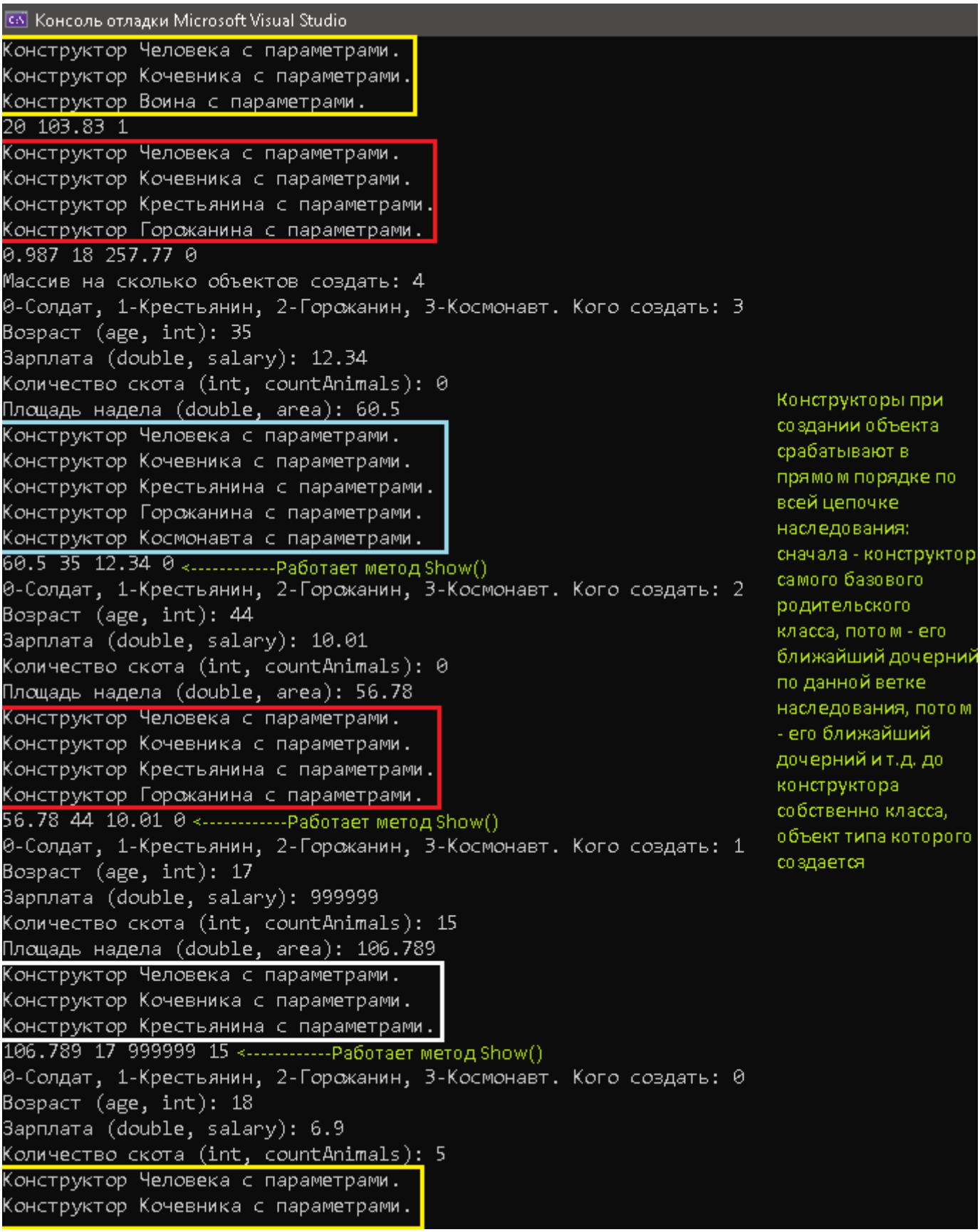

```cpp
#include <iostream>
#include <Windows.h>
#include <time.h>
using namespace std;

class Human//абстрактный класс Человек
{
protected:
    int age = 0;//в классе Человек объявляется целочисленное поле Возраст, которое будет изначально заполнено нулем, чтобы не содержать "пустое мусорное" значение. Конструктор,
    //метод, свойство при этом могут инициализировать данное поле любым другим целочисленным значением, присваивать ему другие значения неограниченное количество раз
    double salary = 0;
public:
    virtual double TakeSalary(double) = 0;//чистый виртуальный метод (ЧВМ) вместо тела {...} имеет код " = 0; ". Класс с хотя бы одним ЧВМ является АБСТРАКТНЫМ, его экземпляры
    //НЕвозможно будет создавать, но можно будет создавать указатели и ссылки типа этого класса, чтобы они указывали на объекты НЕАБСТРАКТНЫХ ЕГО ДОЧЕРНИХ классов, унаследованных
    //от данного класса. Абстрактные классы удобно располагать в начале иерархии наследования для обозначения общих неконкретных понятий и помещения в них таких членов, которые
    //должны быть ВО ВСЕХ остальных классах-наследниках данного абстрактного класса. Абстрактным класс делает наличие в нем хотя бы одного ЧВМ. Если класс-наследник НЕ
    //переопределил с телом все унаследованные им ЧВМ (и которые не переопределены до него его родительскими классами), то он тоже абстрактный класс
    virtual int CelebrateBirthday() = 0;//чистый виртуальный метод
    virtual void Show() = 0;//ЧВМ
    Human(int a, double b) : age(a), salary(b)//параметризованный конструктор с параметрами
    {// age = a; salary = b;
        cout << "Конструктор Человека с параметрами.\n";
    }
    Human() {}//конструктор по умолчанию без параметров
    virtual ~Human()//виртуальный деструктор оюбъектов класса Человек. Если класс содержит ЧВМ, то такой класс должен содержать ВИРТУАЛЬНЫЙ деструктор
    {
        cout << "Деструктор уничтожил объект Человек.\n";
    }
};
class Nomad : public Human//класс Кочевник, наследующий классу Человек, переопределяет оба наследуемых им ЧВМ, иначе останется абстрактным классом
{
protected:
    int countAnimals = 0;
public:
    double TakeSalary(double a)override//метод TakeSalary() виртуальный переопределенный, а уже не ЧВМ, поскольку получает тело в фигурных скобках {...}
    {
        return 0;
```

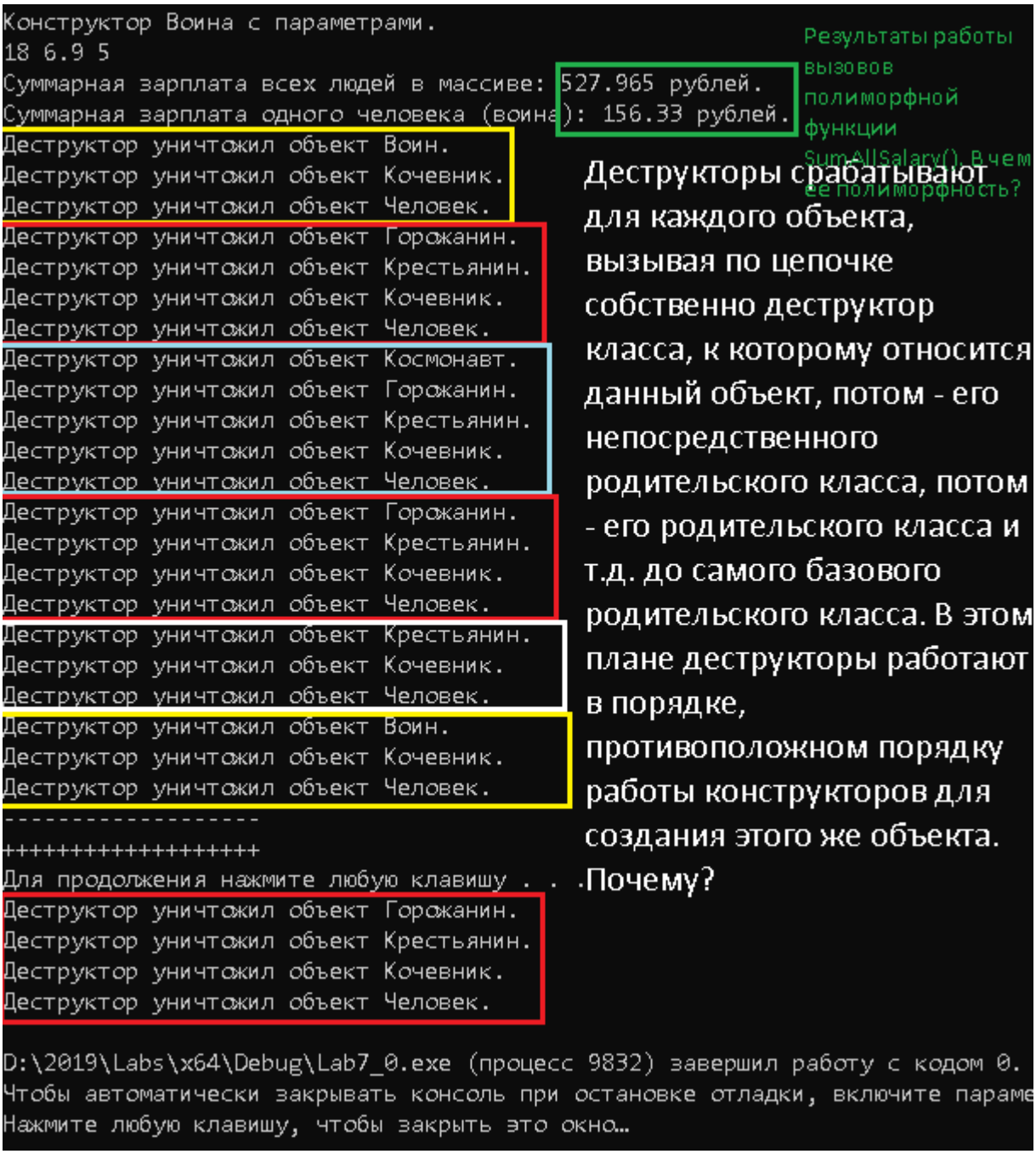
```
    }
    int CelebrateBirthday()override//метод CelebrateBirthday() виртуальный переопределенный, а уже не ЧВМ
    {
        age = age + 1;
        return age;
    }
    void Show()override//метод Show() виртуальный переопределенный, а уже не ЧВМ. Все три унаследованных ЧВМ от абстрактного класса Человек переопределены с телами в этом классе
    {//Кочевник, а потому он является уже обчным классом (а НЕ абстрактным) и МОЖНО создавать экземпляры класса Кочевник
        cout << age << ' ' << salary << ' ' << countAnimals << endl;//аналогично коду: cout << Human::age << Human::salary << Nomad::countAnimals << endl;
    }
    Nomad(int a0, double b0, int c0) : Human(a0, b0), countAnimals(c0)//конструктор с параметрами Nomad() является делегирующим конструктором, т.к. вызывает конструктор
    {//родительского класса Human(int, double) для инициализации двух полей, и параметризованным конструктором, т.к. инициализирует поле countAnimals кодом countAnimals(c0), хотя
        cout << "Конструктор Кочевника с параметрами.\n";//можно было в теле написать аналогичный по результату код: countAnimals = c0;
    }
    Nomad():Human()//конструктор по умолчанию без параметров для класса Кочевник является делегирующим конструктором, так как вызывает конструктор без параметров родительского
    {}//класса Человек
    virtual ~Nomad()//виртуальный деструктор объектов класса Кочевник
    {
        cout << "Деструктор уничтожил объект Кочевник.\n";
    }
};
class Soldier : public Nomad//класс Воин, наследующий классу Кочевник. Уже класс Кочевник переопределил с телами все унаследованные им ЧВМ, а потому наследующий ему класс Воин
{//также является обычным классом, экземпляры (объекты) которого можно создавать. Класс Воин унаследовал переопределенные ВИРТУАЛЬНЫЕ методы, которые он может переопределять
public:
    double TakeSalary(double a)override//метод TakeSalary() виртуальный переопределенный
    {
        srand(time(NULL));
        salary = salary + a + rand() % 50;
        return salary;
    }
    //void Show()override//в классе Воин нет новых полей по сравнению с классом Кочевник, а потому виртуальный метод Show() для него переопределить можно, но в этом НЕТ
    //{// НЕОБХОДИМОСТИ, поскольку ему достаточно унаследованного метода Nomad::Show();, который у него ЕСТЬ ввиду наследования
    //  cout << age << ' ' << salary << ' ' << countAnimals << endl;
    //}
    Soldier(int a1, double b1, int c1) : Nomad(a1, b1, c1)//конструктор с параметрами Soldier() является делегирующим конструктором, поскольку вызывает конструктор родительского
    {//класса Nomad(), чтобы тот проинициализировал поля класса
        cout << "Конструктор Воина с параметрами.\n";
    }
    Soldier():Nomad()//конструктор класса Воин по умолчанию без параметров является делегирующим, так как вызывает конструктор родительского класса Кочевник
    {}
    virtual ~Soldier()//виртуальный деструктор класса Воин
    {
        cout << "Деструктор уничтожил объект Воин.\n";
    }
};
class Peasant : public Nomad//класс Крестьянин (обычный, НЕабстрактный), наследующий классу Кочевник
{
protected:
    double area = 0;
public:
    //double TakeSalary(double a)override//метод TakeSalary() наследуется классом Крестьянин от класса Кочевник, нового тела не содержит, а потому классу Крестьянин достаточно
    //{ унаследованного им метода Nomad::TakeSalary(double);
    //  return 0;
    //}
    void Show()override//переопределяем виртуальный метод Show(), чтобы он печатал значение поля area (родительские классы НЕ имеют такого поля), а значения остальных полей
    {//можно распечатать, вызвав метод Nomad::Show();
        cout << area << ' ';
        Nomad::Show();
    }
    Peasant(int a2, double b2, int c2, double d2) : Nomad(a2, b2, c2), area(d2)//конструктор с параметрами Peasant() является делегирующим конструктором и параметризованным
    {//area = d2;
        cout << "Конструктор Крестьянина с параметрами.\n";
    }
    Peasant():Nomad()//конструктор класса Крестьянин без параметров, делегирующий, так как вызывает для работы конструктор родительского класса Кочевник
    {}
    virtual ~Peasant()//виртуальный деструктор класса Крестьянин
    {
        cout << "Деструктор уничтожил объект Крестьянин.\n";
    }
};
class Citizen : public Peasant//класс Горожанин, наследующий классу Крестьянин
{
public:
    double TakeSalary(double a)override//переопределяем унаследованный метод TakeSalary(double), поскольку хотим, чтобы для объектов ДАННОГО класса Горожанин этот унаследованный
    {//виртуальный метод делал нижеследующий код (у него другое тело - другое поведение)
        salary = salary + a;
        return salary;
    }
    //void Show()override//унаследованного метода Peasant::Show(); классу Горожанин достаточно, так как новых полей в нем не объявлено
    //{
    //  Peasant::Show();
    //}
    Citizen(int a3, double b3, int c3, double d3) : Peasant(a3, b3, c3, d3)//конструктор с параметрами Citizen() является делегирующим конструктором, поскольку вызывает конструктор
    {//родительского класса Крестьянин, чтобы ТОТ проинициализировал все 4 поля, имеющиеся у объекта класса Горожанин
        cout << "Конструктор Горожанина с параметрами.\n";
    }
    Citizen():Peasant()//конструктор без параметров класса Горожанин, является делегирующим конструктором, вызывающим конструктор без параметров родительского класса Крестьянин
    {}
    virtual ~Citizen()//виртуальный деструктор класса Горожанин
    {
        cout << "Деструктор уничтожил объект Горожанин.\n";
    }
};
class Astronaut : public Citizen//класс Космонавт наследует классу Горожанин
{
public:
    double TakeSalary(double a)override//унаследованный виртуальный метод Citizen::TakeSalary(double); переопределяется, так как мы хотим дать ему другое поведение (другое тело)
    {
        salary = 1.5 * salary + a * 3;
        return salary;
    }
    //void Show()override//унаследованного метода Citizen::Show(); классу Космонавт вполне достаточно, поскольку новых полей в нем не объявляется
    //{
    //  Citizen::Show();
    //}
    Astronaut(int a4, double b4, int c4, double d4) : Citizen(a4, b4, c4, d4)//конструктор с параметрами Astronaut() является делегирующим конструктором
    {
        cout << "Конструктор Космонавта с параметрами.\n";
    }
    Astronaut():Citizen()//конструктор без параметров класса Космонавт, делегирующий конструктор, вызывающий конструктор без параметров родительского класса Горожанин
    {}
```

```
    virtual ~Astronaut()//виртуальный деструктор класса Космонавт
    {
        cout << "Деструктор уничтожил объект Космонавт.\n";
    }
};

double SumAllSalary(Human**, int);//прототип полиморфной функции SumAllSalary(), которая может принимать массив указателей на любые объекты из иерархии наследования от абстрактного
//класса Human и полиморфно вызывать у каждого объекта из этой иерархии наследования именно ЕГО виртуальный метод (метод по имени и сигнатуре вызывается один, а работать у каждого
double SumAllSalary(Human** array, int size)//объекта будет метод конкретно его класса. Тут полное определение полиморфной функции, принимающей массив указателей типа класса Human
{//и размер данного массива. Цель этой функции: посчитать суммарнуб зарплату всех людей: кочевников, воинов, крестьян, горожан и космонавтов с учетом того, что зарплата каждому
//начисляется особым образом в зависимости от того, к какому "сословию" он принадлежит: кочевникам и крестьянам зарплата не начисляется совсем (ноль), у воинов зарплатой является
//сумма довольствия (ставка оклада; принимаемое методом входное значение) и случайной добычей (рандом), у горожан и космонавтов зарплаты также расчитываются по отдельным формулам,
//в зависимости от их "сословия". Чтобы правильно посчитать суммарную зарплату всех этих людей, надо знать категорию ("сословие") каждого человека, поскольку каждому начисляется
//зарплата по правилам его "сословия". Благодаря полиморфизму мы напишем универсальный код, который у каждого объекта будет вызывать именно его специализированный метод начисления
//его зарплаты, который зависит от "сословия" человека (конкретного класса, к которому он принадлежит (экземпляром которого он является))
    double summa = 0;//переменная-счетчик для накопления суммарной зарплаты должна быть создана и обнулена
    for (int i = 0; i < size; i++)
    {
        summa += array[i]->TakeSalary(5.5);//проходимся по каждому объекту массива и вызываем у него полиморфный виртуальный переопределяемый метод ПолученияЗарплаты(). Поскольку
    }//метод принимает вещественное значение - ставку зарплаты - передаем ее методу, но благодаря полиморфизму для кочевника и крестьянина она не будет учтена, поскольку отработают
    return summa;//методы конкретных классов, а для объектов других классов вызовутся их специализированные методы, которые будут расчитывать зарплату для объекта по методу именно
}//ЕГО класса. По окончании цикла пошлем из функции суммарную зарплату всех людей, указатели на которых содержал массив

int main()
{
    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);
    //Human a;//ошибка: нельзя создавать объекты типа абстрактного класса Человек ни конструктором без параметров
    //Human a(1, 1.5);//ошибка: нельзя создавать объекты типа абстрактного класса Человек ни конструктором с параметрами
    //Human* a = new Human();//ошибка: нельзя создавать динамические объекты типа абстрактного класса Человек ни конструктором без параметров
    //Human* a = new Human(1, 1.55);//ошибка: нельзя создавать динамические объекты типа абстрактного класса Человек ни конструктором с параметрами
    Human* a = new Soldier(19, 50.96, 1);//можно создать указатель типа абстрактного класса Human, но присваиваем ему динамический объект типа его дочернего НЕабстрактного класса
    //Soldier. Так можно, поскольку конструктор класса Human() НЕ вызывается, а вызывается конструктор НЕабстрактного класса Soldier()
    if (a == nullptr)//проверка на выделение динамической памяти под объект типа класса Soldier
    {
        cout << "Объект типа класса Солдат не удалось создать.\n";
        system("pause");
        return 0;
    }
    a->CelebrateBirthday();//через указатель на объект типа абстрактного класса можно вызывать члены (методы) конкретного объекта его любого дочернего класса
    a->TakeSalary(12.87);
    a->Show();//работает метод Show() класса Soldier, а не абстрактного класса Human
    Citizen b(17, 123.21, 0, 0.987);
    Human& c = b;//создаем ссылку типа абстрактного класса Human и инициализирем ее адресом объекта b дочернего НЕабстрактного класса Citizen
    c.CelebrateBirthday();//через ссылку абстрактного родительского класса Human вызываем методы объекта b дочернего НЕабстрактного класса Citizen
    c.TakeSalary(134.56);
    c.Show();//работает метод Show() класса Citizen, а не абстрактного класса Human
    int n, p, d, e, f, g;//целочисленные переменные для хранения значений, вводимых пользователем
    double h, k;//вещественные переменные для хранения значений, вводимых пользователем
    do {
        cout << "Массив на сколько объектов создать: ";
        cin >> n;
    } while (n < 1);
    Human** mas = new Human*[n];//Human* mas = new Human[n];//ошибка, т.к. вызывается конструктор класса Human() для резервирования памяти, и соответственно, создания объекта типа
    //абстрактного класса Чееловек, что нельзя делать. Можно создать одномерный массив из указателей типа абстрактного класса Human, но поскольку каждый такой указатель - это
    //элемент типа Human*, то массив из них будет массивом второго уровня - Human**, технически как бы двумерный массив. Код = new Human*[n]; создает указатели типа абстрактного
    //класса Human, но не создаются объекты типа класса Human, не вызывается конструктор абстрактного класса Human, что делает такой код допустимым и рабочим, поскольку каждому
    //указателю типа абстрактного родительского класса можно присваивать адреса объектов типа дочерних НЕабстрактных классов из данной иерархии наследования
    if (mas == nullptr)//проверка на выделение динамической памяти под массив указателей
    {
        cout << "Массив указателей типа абстрактного класса Человек не удалось создать.\n";
        system("pause");
        return 0;
    }
    for (int i = 0; i < n; i++)//если массив указателей удалось создать, то можно каждому указателю присвоить адрес любого объекта НЕабстрактного дочернего класса из данной
    {//иерархии наследования
        cout << "0-Солдат, 1-Крестьянин, 2-Горожанин, 3-Космонавт. Кого создать: ";
        cin >> p;//пользователь определяет, объект какого именно НЕабстрактного класса создать в момент выполнения программы
        switch (p)
        {
        case 0:
        {
            cout << "Возраст (age, int): ";//запрашиваем у пользователя значения для инициализации полей объекта НЕабстрактного класса
            cin >> d;
            cout << "Зарплата (double, salary): ";
            cin >> h;
            cout << "Количество скота (int, countAnimals): ";
            cin >> g;
            mas[i] = new Soldier(d, h, g);//создаем динамический объект типа НЕабстрактного класса путем вызова нужного конструктора с параметрами, чтобы сразу поместить в поля
            if (mas[i] == nullptr)//объекта значения пользователя. Делаем проверку на выделение динамической памяти под объект
            {
                cout << "Объект типа класса Воин не удалось создать.\n";
                system("pause");
                return 0;
            }
            break;
        }
        case 1:
        {
            cout << "Возраст (age, int): ";//запрашиваем у пользователя значения для инициализации полей объекта НЕабстрактного класса
            cin >> d;
            cout << "Зарплата (double, salary): ";
            cin >> h;
            cout << "Количество скота (int, countAnimals): ";
            cin >> g;
            cout << "Площадь надела (double, area): ";
            cin >> k;
            mas[i] = new Peasant(d, h, g, k);//создаем динамический объект типа НЕабстрактного класса путем вызова нужного конструктора с параметрами, чтобы сразу поместить в поля
            if (mas[i] == nullptr)//объекта значения пользователя. Делаем проверку на выделение динамической памяти под объект
            {
                cout << "Объект типа класса Крестьянин не удалось создать.\n";
                system("pause");
                return 0;
            }
            break;
        }
        case 2:
        {
            cout << "Возраст (age, int): ";//запрашиваем у пользователя значения для инициализации полей объекта НЕабстрактного класса
            cin >> d;
            cout << "Зарплата (double, salary): ";
```

```
            cin >> h;
            cout << "Количество скота (int, countAnimals): ";
            cin >> g;
            cout << "Площадь надела (double, area): ";
            cin >> k;
            mas[i] = new Citizen(d, h, g, k);//создаем динамический объект типа НЕабстрактного класса путем вызова нужного конструктора с параметрами, чтобы сразу поместить в поля
            if (mas[i] == nullptr)//объекта значения пользователя. Делаем проверку на выделение динамической памяти под объект
            {
                cout << "Объект типа класса Горожанин не удалось создать.\n";
                system("pause");
                return 0;
            }
            break;
        }
        case 3:
        {
            cout << "Возраст (age, int): ";//запрашиваем у пользователя значения для инициализации полей объекта НЕабстрактного класса
            cin >> d;
            cout << "Зарплата (double, salary): ";
            cin >> h;
            cout << "Количество скота (int, countAnimals): ";
            cin >> g;
            cout << "Площадь надела (double, area): ";
            cin >> k;
            mas[i] = new Astronaut(d, h, g, k);//создаем динамический объект типа НЕабстрактного класса путем вызова нужного конструктора с параметрами, чтобы сразу поместить в
            if (mas[i] == nullptr)//поля объекта значения пользователя. Делаем проверку на выделение динамической памяти под объект
            {
                cout << "Объект типа класса Космонавт не удалось создать.\n";
                system("pause");
                return 0;
            }
            break;
        }
        }
        mas[i]->Show();//виртуальный метод Show() вызовется именно у того класса, типа которого будет очередной объект, адрес которого хранит указатель в очередном элементе массива
        mas[i]->CelebrateBirthday();//mas[i]. То есть метод Show() полиморфный - вызывается в коде одинаково по имени,но благодаря позднему связыванию в момент выполнения программы
        mas[i]->TakeSalary(55.66);//вызовется метод того объекта, на который будет указывать указатель. Сам указатель при этом одного типа: абстрактного родительского класса Human
    }
    double sum = SumAllSalary(mas, n);
    cout << "Суммарная зарплата всех людей в массиве: " << sum << " рублей.\n";
    sum = SumAllSalary(&a, 1);
    cout << "Суммарная зарплата одного человека (воина): " << sum << " рублей.\n";
    a->~Human();//динамический объект не нужен - удаляем его с помощью виртуального деструктора, чтобы нужный деструктор для конкретного типа обьекта (класса) "подобрался сам"
    c.~Human();
    for (int i = 0; i < n; i++)//удаляем из динамической памяти двумерный массив
    {
        delete[] mas[i];//деструкторы для каждого объекта вызываются в порядке, ОБРАТНОМ иерархии наследования. Например, для объекта класса Космонавт сначала должен сработать
    }//деструктор класса Космонавт, потом - деструктор класса Горожанин, потом - деструктор класса Крестьянин, потом - деструктор класса Кочевник, потом - деструктор класса Человек
    cout << "--------------------\n";
    delete[] mas;//удаление указателя второго уровня типа абстрактного класса Человек
    cout << "++++++++++++++++++\n";
    system("pause");
    return 0;//по достижении завершающей фигурной скобки функции main() сборщиком мусора удаляются все ранее неудаленные созданные в этой локальной области объекты
}
```

Тестируем программу:


Консоль отладки Microsoft Visual Studio
Конструктор Человека с параметрами.
Конструктор Кочевника с параметрами.
Конструктор Воина с параметрами.
20 103.83 1
Конструктор Человека с параметрами.
Конструктор Кочевника с параметрами.
Конструктор Крестьянина с параметрами.
Конструктор Горожанина с параметрами.
0.987 18 257.77 0
Массив на сколько объектов создать: 4
0-Солдат, 1-Крестьянин, 2-Горожанин, 3-Космонавт. Кого создать: 3
Возраст (age, int): 35
Зарплата (double, salary): 12.34
Количество скота (int, countAnimals): 0
Площадь надела (double, area): 60.5
Конструктор Человека с параметрами.
Конструктор Кочевника с параметрами.
Конструктор Крестьянина с параметрами.
Конструктор Горожанина с параметрами.
Конструктор Космонавта с параметрами.
60.5 35 12.34 0 <----------Работает метод Show()
0-Солдат, 1-Крестьянин, 2-Горожанин, 3-Космонавт. Кого создать: 2
Возраст (age, int): 44
Зарплата (double, salary): 10.01
Количество скота (int, countAnimals): 0
Площадь надела (double, area): 56.78
Конструктор Человека с параметрами.
Конструктор Кочевника с параметрами.
Конструктор Крестьянина с параметрами.
Конструктор Горожанина с параметрами.
56.78 44 10.01 0 <----------Работает метод Show()
0-Солдат, 1-Крестьянин, 2-Горожанин, 3-Космонавт. Кого создать: 1
Возраст (age, int): 17
Зарплата (double, salary): 999999
Количество скота (int, countAnimals): 15
Площадь надела (double, area): 106.789
Конструктор Человека с параметрами.
Конструктор Кочевника с параметрами.
Конструктор Крестьянина с параметрами.
106.789 17 999999 15 <----------Работает метод Show()
0-Солдат, 1-Крестьянин, 2-Горожанин, 3-Космонавт. Кого создать: 0
Возраст (age, int): 18
Зарплата (double, salary): 6.9
Количество скота (int, countAnimals): 5
Конструктор Человека с параметрами.
Конструктор Кочевника с параметрами.
Конструкторы при создании объекта срабатывают в прямом порядке по всей цепочке наследования: сначала - конструктор самого базового родительского класса, потом - его ближайший дочерний по данной ветке наследования, потом - его ближайший дочерний и т.д. до конструктора собственно класса, объект типа которого создается



3.Порядок выполнения работы

1. Изучить теоретические сведения к лабораторной работе (читай выше).
2. Реализовать алгоритм решения задачи.

Задание 3.

Написать и протестировать программу:

Напишите *абстрактный* класс HOMOSAPIENS, которому будет наследовать дочерний класс ЧЕЛОВЕК (PERSON).

Базовый класс:

ЧЕЛОВЕК (PERSON) имеет защищенные (protected) поля:

Имя (name) – char[15],

Возраст (age) – int.

Определить методы установки (изменения) и возврата значений, хранимых в полях класса (то есть свойства – геттеры и сеттеры).

Создать производный (дочерний) класс СОТРУДНИК (EMPLOYEE), имеющий также поля:

Должность – char[30],

Оклад – double.

Определить методы изменения полей и вычисления зарплаты сотрудника по формуле Оклад+Премия (премия – это процент, на который увеличивается месячный оклада сотрудника; значение процента метод получает входным вещественным параметром). Создать в функции main() объекты классов и продемонстрировать их работу.

3. Разработать на языке C++ программу вывода на экран решения задачи в соответствии с вариантом индивидуального задания, указанным преподавателем.

4. Отлаженную, работающую программу сдать преподавателю. Работу программы показать с помощью самостоятельно разработанных тестов.

5. Ответить на контрольные вопросы.

Задание 4.

Выполните задачу по своему варианту. Если потребуется, адаптируйте свое задание. Например, чтобы создавать объекты типа классов Точка и т.д., нужно, чтобы данные классы были **НЕ**абстрактными, соответственно, выгодно унаследовать класс Точка от абстрактного класса Element.

№	Задача
1	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового, разработать производный класс Circle, определяющий окружности разного радиуса. Определить в этом классе метод, возвращающий расстояние между центрами двух окружностей. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на объекты точек и окружностей. Вывести среднее расстояние между центрами окружностей.
2	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового, разработать производный класс Circle, определяющий окружности разного радиуса. Определить в этом классе методы, возвращающие длину окружности и площадь круга. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на объекты точек и окружностей. Вывести среднее арифметическое длин окружностей и среднее арифметическое их площадей.
3	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового, разработать производный класс Circle, определяющий окружности разного радиуса. Определить в этом классе метод, возвращающий расстояние между центрами двух окружностей. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на объекты точек и окружностей. Вывести среднее расстояние между центрами окружностей.
4	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового, разработать производный класс Circle, определяющий окружности разного радиуса. Определить в этом классе метод, возвращающий расстояние между центром окружности и началом координат. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на объекты и точек и окружностей. Вывести среднее расстояние от центров окружностей до начала координат.

5	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового, разработать производный класс Circle, определяющий окружности разного радиуса. Определить в этом классе метод, возвращающий площадь треугольника, вершинами которому служат центры трех заданных окружностей. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на объекты и объявить несколько точек и 3 окружности. Вывести площадь треугольника, вершинами которого являются центры заданных окружностей.
6	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового разработать производный класс Circle, определяющий окружности разного радиуса. Определить в этом классе метод, возвращающий длину радиуса окружности, описанной вокруг треугольника, вершинами которому служат центры трех заданных окружностей. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на объекты и создать несколько точек и 3 окружности. Вывести длину радиуса окружности, описанной вокруг треугольника, вершинами которого являются центры заданных окружностей.
7	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового, разработать производный класс Circle, определяющий окружности разного радиуса. Определить в этом классе метод, возвращающий минимальное расстояние от начала координат до окружности. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на объекты и создать несколько точек и несколько окружностей. Вывести минимальное значение расстояния от начала координат до окружностей.
8	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового, разработать производный класс Circle, определяющий окружности разного радиуса. В обоих классах объявить виртуальную метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек и 3 окружности. Определить в этом классе метод, возвращающий длины медиан треугольника, вершинами которому служат центры трех заданных окружностей и вывести эти длины в главной программе.
9	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового, разработать производный класс Circle, определяющий окружности разного радиуса. Определить в этом классе метод, возвращающий длины биссектрис треугольника, вершинами которому служат центры трех заданных окружностей. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек и 3 окружности. Вывести длины биссектрис.
10	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового, разработать производный класс Circle, определяющий окружности разного радиуса. Определить в этом классе метод, возвращающий длину радиуса окружности, вписанной в треугольник, вершинами которому служат центры трех заданных окружностей. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек и 3 окружности. Вывести длину вписанной окружности.
11	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового класса, разработать производный класс Rectangle, определяющий различные прямоугольники со сторонами, параллельными осям координат. Определить в этом классе метод, возвращающий координаты всех вершин прямоугольника. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек и несколько прямоугольников. Вывести центр масс всех вершин прямоугольников.

12	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового класса, разработать производный класс Rectangle, определяющий различные прямоугольники со сторонами, параллельными осям координат. Определить в этом классе метод, возвращающий длину окружности, описанной вокруг данного прямоугольника. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек и несколько прямоугольников. Вывести длину окружности, описанной вокруг первого прямоугольника.
13	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового класса, разработать производный класс Rectangle, определяющий различные прямоугольники со сторонами, параллельными осям координат. Определить в этом классе метод, возвращающий площадь прямоугольника. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек и несколько прямоугольников. Вывести сумму площадей всех прямоугольников.
14	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового класса, разработать производный класс Rectangle, определяющий различные прямоугольники со сторонами, параллельными осям координат. Определить в этом классе метод, возвращающий площадь круга, описанного вокруг данного прямоугольника. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек и несколько прямоугольников. Вывести сумму площадей кругов, описанных вокруг прямоугольников.
15	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового класса, разработать производный класс Rectangle, определяющий различные прямоугольники со сторонами, параллельными осям координат. Определить в этом классе метод, возвращающий расстояния от начала координат до всех вершин прямоугольника. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек и несколько прямоугольников. Вывести среднее расстояние от начала координат до вершин прямоугольников.
16	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового класса, разработать производный класс, определяющий различные прямоугольники со сторонами, параллельными осям координат. Определить в этом классе метод, возвращающий площадь четырехугольника, вершинами которого служат середины сторон данного прямоугольника. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек и несколько прямоугольников. Вывести площадь вписанных четырехугольников.
17	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового класса, разработать производный класс, определяющий различные прямоугольники со сторонами, параллельными осям координат. Определить в этом классе логический метод, определяющий принадлежность точки (принимается методом) данному прямоугольнику. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек и несколько прямоугольников. Вывести сколько точек находится внутри прямоугольников.

18	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового, разработать производный класс Circle, определяющий окружности разного радиуса. Определить в этом классе логический метод, определяющий принадлежность точки (принимается методом) к данной окружности. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массивы указателей на несколько точек и 3 окружности. Вывести сколько точек находится внутри окружностей.
19	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового, разработать производный класс Circle, определяющий окружности разного радиуса. Новый класс использовать в качестве базового для разработки класса Cylinder, определяющий различные цилиндры. Определить в этом классе метод, возвращающий объем цилиндра. Во всех классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек, 2 окружности и 3 цилиндра. Вывести средний объем цилиндров.
20	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового, разработать производный класс Circle, определяющий окружности разного радиуса. Новый класс использовать в качестве базового для разработки класса Cylinder, определяющий различные цилиндры. Определить в этом классе метод, возвращающий площадь полной (всей) поверхности цилиндра. Во всех классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек, 3 окружности и 2 цилиндра. Вывести суммарную площадь поверхности цилиндров.
21	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового класса, разработать производный класс Polygon, определяющий правильные многоугольники. Определить в этом классе метод, определяющий площадь данного многоугольника. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек и 3 многоугольника. Вывести среднюю площадь многоугольников.
22	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового класса, разработать производный класс Polygon, определяющий правильные многоугольники. Определить в этом классе метод, определяющий радиус окружности, описанной вокруг данного многоугольника. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек и 3 многоугольника. Вывести среднюю длину радиусов окружностей.
23	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового класса, разработать производный класс Polygon, определяющий правильные многоугольники. Определить в этом классе метод, определяющий радиус окружности, вписанной в данный многоугольник. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек и 3 многоугольника. Вывести среднюю длину радиусов окружностей.
24	Разработать класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового, разработать производный класс Circle, определяющий окружности разного радиуса. Новый класс использовать в качестве базового для разработки класса Cone, определяющий различные конусы. Определить в этом классе метод, возвращающий площадь полной поверхности конуса. Во всех классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек, 2 окружности и 3 конуса. Вывести среднюю площадь поверхности конусов.

25	Разработать класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового, разработать производный класс Circle, определяющий окружности разного радиуса. Новый класс использовать в качестве базового для разработки класса Cone, определяющий различные конусы. Определить в этом классе метод, возвращающий объем конуса. Во всех классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек, 2 окружности и 3 конуса. Вывести суммарный объем конусов.
26	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового класса, разработать производный класс, определяющий различные прямоугольники со сторонами, параллельными осям координат. Определить в этом классе логический метод, определяющий принадлежность точки (принимается методом) данному прямоугольнику. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек и несколько прямоугольников. Вывести сколько точек находится внутри прямоугольников.
27	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового класса, разработать производный класс Rectangle, определяющий различные прямоугольники со сторонами, параллельными осям координат. Определить в этом классе метод, возвращающий координаты всех вершин прямоугольника. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек и несколько прямоугольников. Вывести центр масс всех вершин прямоугольников.
28	Разработать абстрактный класс Point для задания координаты точки на плоскости. Выбирая этот класс в качестве базового, разработать производный класс Circle, определяющий окружности разного радиуса. Определить в этом классе метод, возвращающий длину радиуса окружности, вписанной в треугольник, вершинами которому служат центры трех заданных окружностей. В обоих классах объявить виртуальный метод Length, который возвращает длину (периметр) соответствующего объекта. В главной программе объявить массив указателей на несколько точек и 3 окружности. Вывести длину вписанной окружности.

4. 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? Приведите примеры.
4. В каких случаях используется механизм позднего связывания?

Литература

Страуструп, Б. Язык программирования C++ / Б. Страуструп. – СПб. : БИНОМ, 2011.

Павловская, Т. А. C++. Объектно-ориентированное программирование: практикум / Павловская, Т. А., Щупак. – СПб. : Питер, 2011.

Преподаватель Белокопыцкая Ю.А.

Рассмотрено на заседании цикловой
комиссии ПОИТ № 10
Протокол №____от «___» _____2017 г.
Председатель ЦК _____