КУРСОВАЯ РАБОТА

Наследование

1 Цель и порядок работы

Цель работы – получение практических навыков по созданию и использованию принципов наследования в языке C# при построении математических абстракций. Максимальное использование наследования, даже если для конкретной задачи оно не дает выигрыша в объеме программы.

Порядок выполнения работы:

- ознакомиться с описанием лабораторной работы;
- получить задание у преподавателя, согласно своему варианту;
- написать программу и отладить ее на ЭВМ;
- оформить отчет.

Теоретические сведения

Теоретический материал по лабораторной работе приведен в:
1) соответствующем лекционном материале;
2) работах [1-3, 5, 7] из списка литературы;
3) http://msdn.microsoft.com/ru-ru/visualc/default.aspx
4) В разделе «Help» меню Visual Studio 2017.

2. Теоретическая часть

Класс является обобщенным понятием, определяющим характеристики и поведение некоторого множества конкретных объектов этого класса, называемых экземплярами, или объектами, класса.

В целях ускорения работы программиста и упрощения применяемых алгоритмов применяют классы. Применение классов позволяет разбить сложную задачу на ряд простых, взаимосвязанных задач. Проделав эту операцию неоднократно применительно к вновь полученным классам, можно получить алгоритм решения задачи в виде набора простых, понятных, легко осуществимых классов, взаимодействующих друг с другом.

«Классический» класс содержит данные, задающие свойства объектов класса, и функции, определяющие их поведение. В последнее время в класс часто добавляется третья составляющая - события, на которые может реагировать объект класса. Это оправдано для классов, использующихся в программах, построенных на основе событийно-управляемой модели, например, при программировании для Windows.

Процедура объявления и создания экземпляров пользовательского типа не отличается от таковой для предопределенных типов за исключением необходимости использования Спецификатора **New** (**new**) для создания экземпляров пользовательских типов значений и ссылочных типов.

Спецификаторы классов**:**

new - используется для вложенных классов. Задает новое описание класса взамен унаследованного от предка. Применяется в иерархиях объектов;

public - доступ не ограничен**;**

protected - используется для вложенных классов. Доступ только из элементов данного и производных классов**;**

internal - доступ только из данной программы (сборки)**;**

protected internal - доступ только из данного и производных классов или из данной программы (сборки)**;**

private - используется для вложенных классов. Доступ только из элементов класса, внутри которого описан данный класс**;**

abstract - абстрактный класс. Применяется в иерархиях объектов**;**

sealed - бесплодный класс. Применяется в иерархиях объектов**;**

static - статический класс. Введен в версию языка 2.0.

Спецификаторы называются спецификаторами доступа. Они определяют, откуда можно непосредственно обращаться к данному классу. Спецификаторы доступа могут присутствовать в описании только в вариантах, приведенных в таблице, а также могут комбинироваться с остальными спецификаторами.

Объявление **класса** состоит из нескольких элементов:

- объявление атрибутов[1] – необязательный элемент объявления,
- модификаторы (в том числе модификаторы прав доступа) – необязательный элемент объявления,
- partial (спецификатор разделения объявления класса) – необязательный элемент объявления,
- class,
- имя класса,
- имена предков (класса и интерфейсов) – необязательный элемент объявления,
- тело класса.

Модификаторы прав доступа (см. табл.1.) обеспечивают реализацию принципа инкапсуляции, используются при объявлении классов и их составляющих компонентов.

Табл. 1. Модификаторы прав доступа членов класса.

1	2
public	обозначение для общедоступных членов класса. К ним можно обратиться из любого метода любого класса программы.

[1] Атрибуты – средство добавления ДЕКЛАРАТИВНОЙ (вспомогательной) информации к элементам программного кода. Назначение атрибутов: организация взаимодействия между программными модулями, дополнительная информация об условиях выполнения кода, управление сериализацией (правила сохранения информации), отладка, многое другое.

protected	обозначение для членов класса, доступных в рамках объявляемого класса и из методов производных классов.

Окончание Табл. 1.

1	2
internal	обозначение для членов класса, доступных из методов классов, объявляемых в рамках сборки, содержащей объявление данного класса.
protected internal	обозначение для членов класса, доступных в рамках объявляемого класса и из методов производных классов, а также доступных из методов классов, объявляемых в рамках сборки, содержащей объявление данного класса.
private	обозначение для членов класса, доступных в рамках объявляемого класса.

В теле **класса** могут быть объявлены: **константы, поля, конструкторы и деструкторы, методы, свойства, делегаты, классы.**

О свойствах и делегатах предстоит подробный разговор в последующих лабораторных. Из синтаксиса следует, что классы могут быть вложенными. Такая ситуация довольно редкая. Ее стоит использовать, когда некоторый класс носит вспомогательный характер, разрабатывается в интересах другого класса и есть полная уверенность, что внутренний класс никому не понадобится кроме класса, в который он вложен. Как уже упоминалось, внутренние классы обычно имеют модификатор доступа, отличный от public. Основу любого класса составляют его конструкторы, поля и методы.

Поля класса

Поля класса синтаксически являются обычными переменными (объектами) языка Их описание удовлетворяет обычным правилам объявления переменных. Поля задают представление той самой абстракции данных, которую реализует класс. Поля характеризуют свойства объектов класса. Два объекта одного класса имеют один и тот же набор полей, но разнятся значениями, хранимыми в этих полях.

Методы

Методы делают всю работу, для исполнения которой предназначены классы и структуры: вычисляют значения, изменяют данные, получают вводимую информацию - в общем, выполняют все действия, составляющие поведение объекта

Синтаксически в описании метода различают две части – описание заголовка и описание тела метода:

```
заголовок_метода
{
    тело_метода
}
```

Синтаксис заголовка метода:

[атрибуты][модификаторы]{void|тип_результата_функции} имя_метода([список_формальных_аргументов])

Имя метода и список формальных аргументов составляют сигнатуру метода. Заметьте, в сигнатуру не входят имена формальных аргументов, здесь важны типы аргументов. В сигнатуру не входит и тип возвращаемого результата. Модификаторы уровня доступа к методу определяют откуда можно будет обратиться к данному методу (табл. 2).

Табл. 2. Модификаторы уровня доступа.

Модификатор	Назначение
public	Метод открыт и доступен для вызова клиентами и потомками класса
private	Метод предназначен для внутреннего использования в классе и доступен для вызова только в теле методов самого класса
protected	Метод будет доступен как из класса, в котором он определен, так и из любого производного класса. Для остальных вызовов из вне этот метод будет недоступен.
internal	Метод будет доступен из всех классов внутри сборки, в которой он определен. Из-за пределов этой сборки обратиться к нему будет нельзя.
protected internal	Действует как protected или как internal

```
// Уровни доступа к методам
class SomeClass
(
// Доступен отовсюду
public void MethodA(){};
// Доступен только из типов данных SomeClass
private void MethodB(){};
// Доступен только из SomeClass и из классов, производных от SomeClass
// (на любом нижестоящем уровне иерархии)
protected void MethodC(){};
// Доступен только из той же самой сборки
internal void MethodD(){};
// Будет действовать как protected или internal
```

```
protected internal void MethodE(){};
// Будет считаться protected - по умолчанию
void MethodO {};
}
```

Обязательным при описании заголовка является указание типа результата, имени метода и круглых скобок, наличие которых необходимо и в том случае, если сам список формальных аргументов отсутствует. Формально тип результата метода указывается всегда, но значение void однозначно определяет, что метод реализуется процедурой, т.е. не возвращает никакого результата и не использует return. Тип результата, отличный от void, указывает на функцию.

Вот несколько простейших примеров описания методов:

```
void A() {…};
int B(){…};
public void C(){…};
```

Методы A и B являются закрытыми, а метод C – открыт. Методы A и C реализованы процедурами, а метод B – функцией, возвращающей целое значение.

Методы могут быть объявлены как статические — с использованием ключевого слова static. Это значит, что статический метод (static method) может быть вызван напрямую через уровень класса, без необходимости создавать хотя бы один экземпляр объекта данного класса. По этой причине метод Main() всегда объявляется как static — чтобы этот метод мог начать выполняться еще до создания первого экземпляра класса, в котором он определен.

Объявление методов

Один из видов членов, которые добавляются к классам, - это методы, определяющие действия, которые должен выполнять класс. Есть две разновидности методов: те, что возвращают значения, и те, которые никаких значений не возвращают. Следующий пример иллюстрирует обе разновидности методов:

```
// Этот метод возвращает целые (int) значения public
int Add (int first, int second)
{
int Result;
Result = first + second;
Return result;
}
```

```
public void myVoidMethod ()
// void - не возвращает значений
{
MessageBox.Show ("Этот метод не возвращает значения ");
}
```

Метод является самостоятельной частью кода, которая имеет имя и может содержать аргументы, выполнять последовательность инструкций (операторов) и изменять значения своих аргументов.

Вызов методов.

Метод не выполняется, пока его не вызовут. Чтобы вызвать метод, следует указать его имя и задать необходимые ему параметры.

// Так вызывают метод Rotate с двумя параметрами Rotate (45, "Degrees");

Метод Main - особый, он вызывается сразу после начала исполнения программы. К особым методам также относятся деструкторы, которые вызывают во время выполнения непосредственно перед уничтожением объекта. Третий вид особых методов - конструкторы; они исполняются во время инициализации объектов.

Метод может иметь один или несколько параметров. Параметр — это аргумент, передаваемый методу вызывающим его методом. При объявлении метода список параметров указывают после его имени в круглых скобках, при этом для каждого параметра необходимо указать его тип.

```
public void DisplayName (string name, byte age)
{
Console.WriteLine ("Hello " + name + ". You are " +
age.ToString () +"years old.");
}
```

У этого метода два параметра: один типа String, с локальным именем name, а второй, age, - типа Byte. Их область видимости ограничена методом, вне которого переменные не доступны.

Пример .1. Задан вектор D(10). Создать класс vect осуществляющий ввод данных, вывод элементов вектора на консоль, и перестановку первого по порядку элемента с k-тым отрицательным элементом вектора. Ограничение доступа в класс (снаружи можно вызывать только public части класса)

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        class vect //  класс векторов vect
```

```
        {
            double[] a;      // объявление массива
            int n;     //   размер массива
            public vect(int n)
            // конструктор создаёт вектор
            {
                this.a = new double[n];
                this.n = n;
            }
            public int vv()
            // заполнение вектора целиком,
            // клавиатурный ввод
            {
                Random rnd = new Random();
                for (int i = 0; i < n; i++)
            {
            // заполнение строки
            a[i] = -50+rnd.Next(1, 101);
                    // преобразование строки в число
                }
                return (0);
    }
            public int vyv()
            // вывод на консоль
            // элементов вектора
            {
                for (int i = 0; i < n; i++)
                    Console.WriteLine("a[" + i + "]=" +
a[i]);
                return (0);
            }
            public int perest(int k)
            // переставить 1 и
            // k-ое отриц. число
            {
                // ключ; 0 если порядок
                // -1,  если - нет
                int kl = 0;
                // к больше размера вектора
                if (k >= n) kl = -1;
                else
                {
                    // счетчик отрицательных чисел
                    int cnt = 0;
```

```
                    // указатель на позицию в массиве
                    int pos = -1;
                    double tmp;
                    // подсчёт отрицательных чисел
                    while (cnt < k)
                    {
                        pos++;
                        if (a[pos] < 0) cnt++;
                    }
                    // подсчёт
                    tmp = a[pos];
                    // перестановка
                    a[pos] = a[0];
                    a[0] = tmp;
                    kl = 0;
                }
                return kl;
            }
            // главная программа
            static void Main()
            {                                   // создали
                vect d = new vect(10);
                d.vv();// заполнили
                d.vyv();// вывели на консоль
                Console.WriteLine("\n--------------------");
                Console.WriteLine("Перестановочный
результат");
                d.perest(3); //переставили
                d.vyv();//вывели на консоль
                Console.ReadKey();
            }
        }
    }
}
```

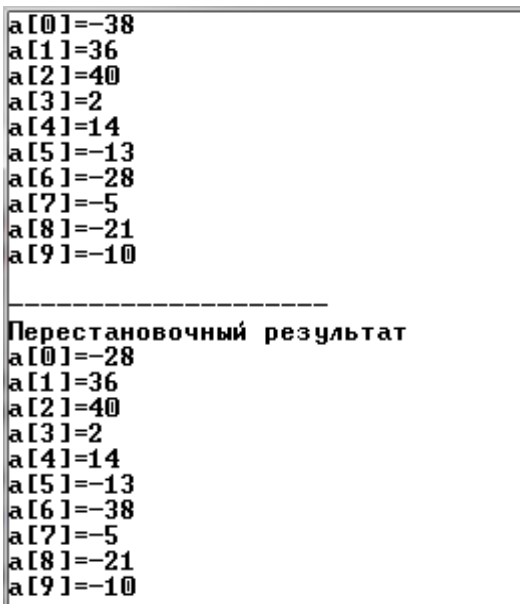


Рис.1 Листинг примера 1

Пример 2. Задана матрица D(3,3). Написать метод транспонирования матрицы и метод перемножения квадратных матриц. Получить результат умножения транспортированной матрицы на соответствующую ей исходную. Отдельный класс для матриц не создаём. Работаем в классе myprog. Поэтому все содержимое класса доступно.

```
using System;
// моё пространство имён
  namespace ConsoleApplication3
        {
            class myprog
            {
                static void Main()
                {
                    // объявление матриц
                    int n = 3;
                    double[,] d = new double[n, n];
                    double[,] dt = new double[n, n];
                    double[,] ddt = new double[n, n];
                    // заполнение d с клавиатуры
                    int i; int j;
                    // строки, колонки
                    for (i = 0; i < n; i++)
                    {
                        for (j = 0; j < n; j++)
                            d[i, j] =
Convert.ToDouble(Console.ReadLine());
```

```
                    }
                    // вывод на консоль
                    // элементов вектора
                    Console.Write("исходная");
                    for (i = 0; i < n; i++)
                    {
                        Console.WriteLine();
                        for (j = 0; j < n; j++)
                        {
                            Console.Write("{0, 5}", d[i,
j]);
                        }
                    }
                    Console.WriteLine();
                    // траспонирование матрицы
                    transp(d, n, dt);
                    // умножение транпонированной
                    // матрицы dt на исходную d
                    mt_q_mult(dt, d, ddt, n);
                    // транпонирование
                }
    static  void transp(double[,] mtr, int p, double[,]
mtrt)
    // double[,] mtr - так определяют матрицу в
     // формальных параметрах
  {
        int i;    int j;
        Console.Write("транпонирование");
        for (i    = 0; i   <    p;   i++)
        {
            Console.WriteLine();
            for (j = 0; j < p; j++)
            {
                mtrt[i, j] = mtr[j, i];
                Console.Write("{0, 5}", mtrt[i, j]);
            }
        }
 }// умножение квадратных матриц
 static void mt_q_mult(double[,] m_a, double[,] m_b,
double[,] m_r, int p) // m_r=m_a * m_b , int p размер матриц
    {
        int i;    int j;   int k;
        double    mel;
        Console.WriteLine();
```

```
        Console.Write("умножение квадратных матриц");
        for (i    = 0; i   <    p;   i++) //  строки m_a
             {
                 Console.WriteLine(); // столбцы m_b
                 for (j = 0; j < p; j++)
                 {
                    mel=0; // столбцы m_a и строки m_b
                    for (k = 0; k < p; k++)
                        {
                          mel = mel + m_a[i, k] * m_b[k, j];
                          m_r[i, j] = mel;
                        }
                    Console.Write("{0, 5}", m_r[i, j]);
                 }
             }
        Console.ReadKey();
    }
}
}
```

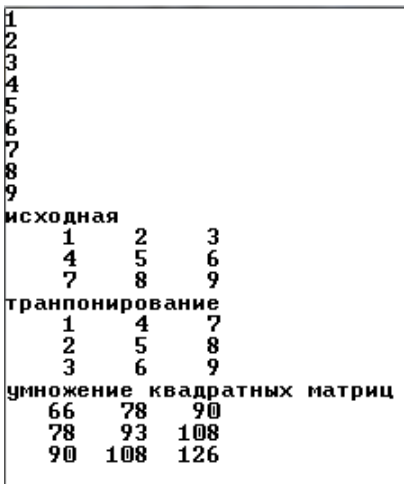
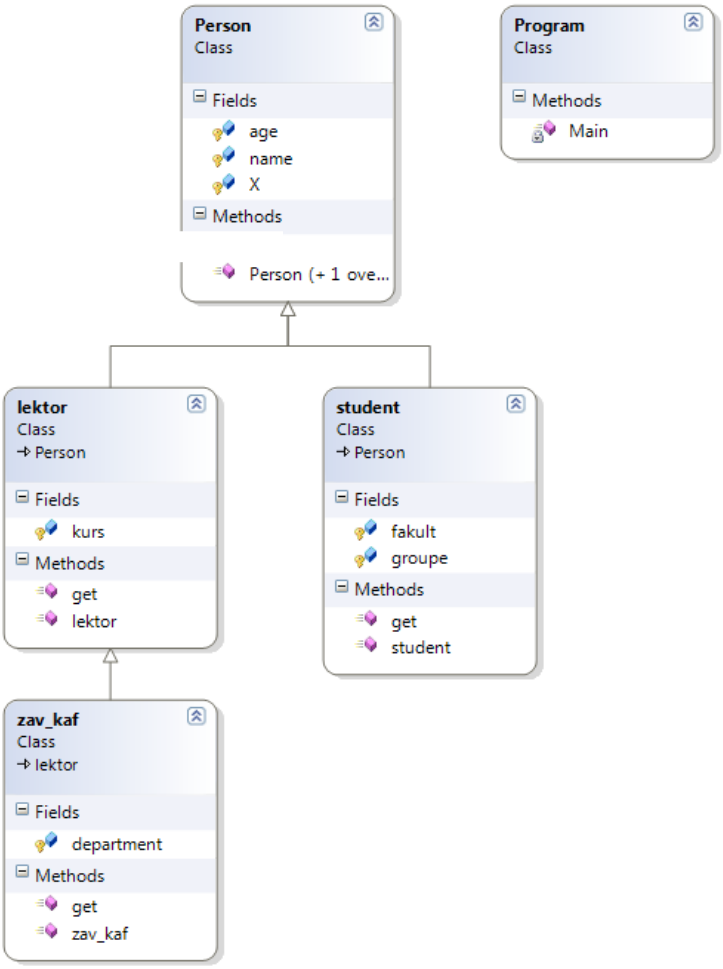


Рис.2 Листинг примера 2

Пример 3. Построить иерархию классов в соответствии с вариантом задания:

1) Студент, преподаватель, персона, заведующий кафедрой

Иерархия классов



Клас Person – Описывает персону.
Клас student – Описывает студента. Производный от Person
Клас lector – описывает преподавателя. Производный от Person.
Клас zav_kaf – описывает заведующего кафедры. Производный от lector.

Листинг 3

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ID
{//--------класс персона-----------------
    class Person
    {
        protected string name;
        protected int age;
        public Person()
        {
            Console.WriteLine("Name?");
            this.name=Console.ReadLine();
            Console.WriteLine("Age?");
            this.age = Convert.ToInt32(Console.ReadLine());
        }
        public virtual string get()
        {
            return String.Format("Person : Name - {0}, Age -
{1}", this.name, this.age);
        }
    }
    //--------класс студент------------
    class student : Person
    {
        protected string groupe;
        protected string fakult;
        public student()
        {
            this.groupe = "zik81";
            this.fakult = "FIOT";
        }
        public override string get()
        {
            return String.Format("Student : Name - {0}, Age
- {1} Fakult - {2}, Group - {3}", this.name, this.age,
this.fakult, this.groupe);
        }
    }
    //----------класс лектор-----------------
    class lektor : Person
    {
```

```
        protected string kurs;
        public lektor()
        {
            this.kurs = "phisycs";
        }
        public override string get()
        {
            return String.Format("Lecturer : Name - {0}, Age
- {1}, Kourse - {2}", this.name, this.age, this.kurs);
        }
    }
    //-----класс заведующий кафедрой-------------
    class zav_kaf : lektor
    {
        protected string department;

        public zav_kaf()
        {
            this.department = "TK";
        }
        public override string get()
        {
            return String.Format("Zav_kaf : Name - {0}, Age
- {1}, Kourse - {2}, Department- {3}", this.name, this.age,
this.kurs, this.department);
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter Student Info");
            Person B = new student();
            Console.WriteLine("Enter Lector Info");
            Person C = new lektor();
            Console.WriteLine("Enter Zav Info");
            Person D = new zav_kaf();
            Console.WriteLine(B.get());
            Console.WriteLine(C.get());
            Console.WriteLine(D.get());
            Console.ReadKey();
        }
    }
}
```

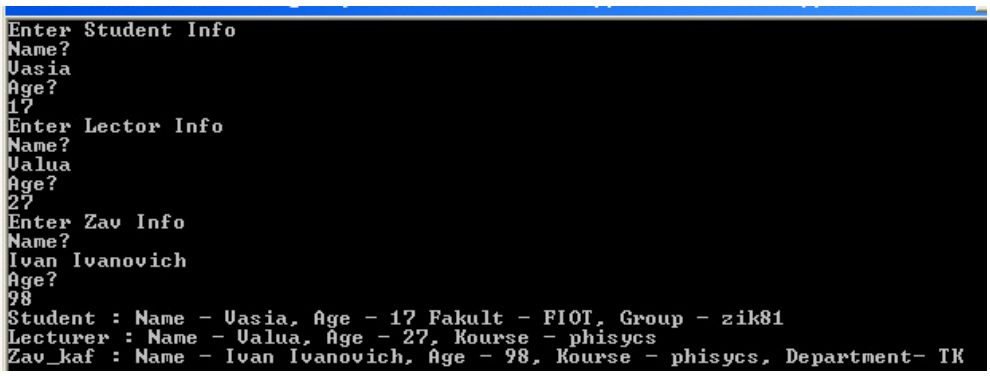
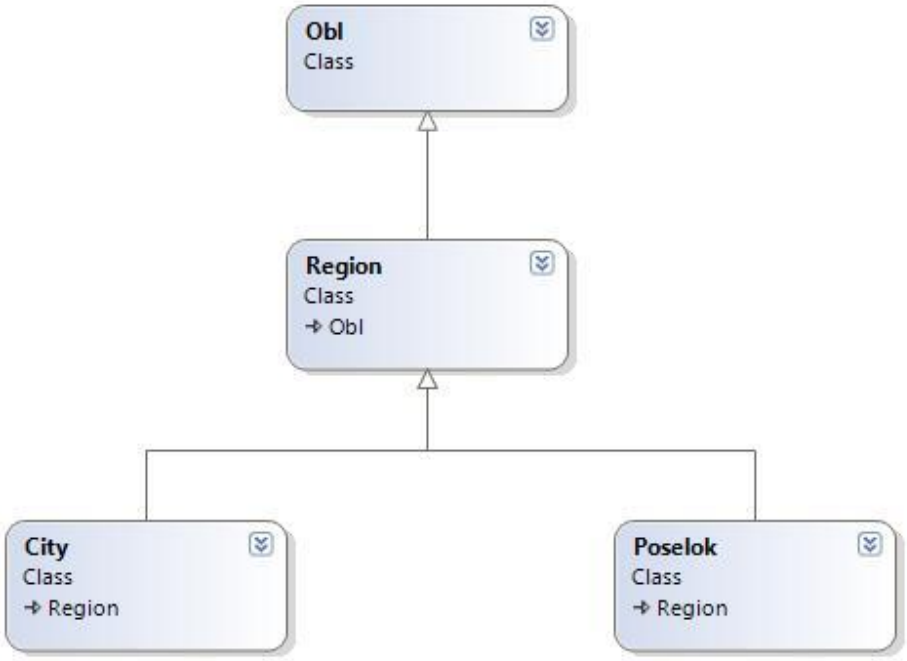


Рис.3 Листинг примера 3

Пример 4. Реализовать программу в C#, в которой используются классы село, область, район, город.

Иерархия классов

Клас Obl –Описывает область.



Клас Region – Описывает Регион. Производный от Obl
Клас City – Описывает город. Производный от Region.
Клас Poselok – Описывает село. Производный от Region.

Код программы на языке C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication5
{
```

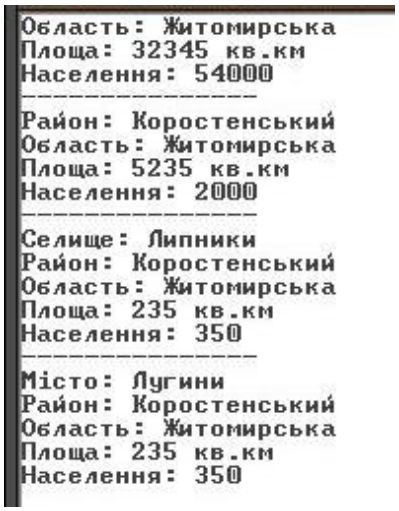
```
    class Obl
    {
        public Obl()
        {
            oName = "xz";
            Population = 12;
            Square = 5;
        }
        public Obl(string oName, int Population, float
Square)
        {
            this.oName = oName;
            this.Population = Population;
            this.Square = Square;
        }
        protected string oName;
        protected int Population;
        protected float Square;
        public string retOname()
        {
            return oName;
        }
        public string retPop()
        {
            return Convert.ToString(Population);
        }
        public string retSq()
        {
            return string.Format("{0} кв.км", Square);
        }
        public string PrintAll()
        {
            return string.Format("Область: {0}\nПлоща:
{1}\nНаселення: {2}", oName, retSq(), Population);
        }
    }
    class Region : Obl
    {
        public Region()
        {
            regName = "neva#no";
        }
        public Region(string oName, string regName, int
Population, float Square)
```

```
        {
            this.oName = oName;
            this.Population = Population;
            this.Square = Square;
            this.regName = regName;
        }
        protected string regName;
        public string retRname()
        {
            return regName;
        }
        public string PrintAll()
        {
            return string.Format("Район: {0}\nОбласть:
{1}\nПлощадь: {2}\nНаселение: {3}", regName, oName, retSq(),
Population);
        }
    }
    class Poselok : Region
    {
        public Poselok()
        {
            posName = "PosNeva#no";
        }
        public Poselok(string oName, string regName, string
posName, int Population, float Square)
        {
            this.oName = oName;
            this.Population = Population;
            this.Square = Square;
            this.regName = regName;
            this.posName = posName;
        }
        protected string posName;
        public string retPosName()
        {
            return posName;
        }
        public string PrintAll()
        {
            return string.Format("Село: {0} \nРайон:
{1}\nОбласть: {2} \nПлощадь: {3} \nНаселение: {4}", posName,
regName, oName, retSq(), Population);
        }
```

```
    }
    class City : Region
    {
        public City()
        {
            cName = "Cityname";
        }
        public City(string oName, string regName, string
cName, int Population, float Square)
        {
            this.oName = oName;
            this.Population = Population;
            this.Square = Square;
            this.regName = regName;
            this.cName = cName;
        }
        protected string cName;
        public string retCname()
        {
            return cName;
        }
        public string PrintAll()
        {
            return string.Format("Город: {0}\nРайон:
{1}\nОбласть: {2} \nПлощадь: {3} \nНаселение: {4}", cName,
regName, oName, retSq(), Population);
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Obl Obl1 = new Obl("Житомирская", 54000, 32345);
            Region reg1 = new Region("Житомирская",
"Коростенский", 2000, 5235);
            Poselok pos1 = new Poselok("Житомирская",
"Коростенский", "Липники", 350, 235);
            City city1 = new City("Житомирская",
"Коростенский", "Лугини", 350, 235);
            Console.WriteLine(Obl1.PrintAll());
            Console.WriteLine("---------------");
            Console.WriteLine(reg1.PrintAll());
            Console.WriteLine("---------------");
            Console.WriteLine(pos1.PrintAll());
```

```
            Console.WriteLine("---------------");
            Console.WriteLine(city1.PrintAll());
            Console.ReadKey();
        }
    }
}
```

Результат работы программы



3. Задание

1. Спроектировать абстракции и представить иерархию классов в виде схемы.
2. Разработать конструкторы, атрибуты и методы для каждого из определяемых классов.
3. Реализовать программу на C# в соответствии с вариантом исполнения, в которой используются экземпляры описанных классов.
4. Применить и объяснить необходимость использования принципа инкапсуляции.
5. Подготовить документальный отчет, содержащий пять документов:
   5.1. Таблица «Наименование сервиса (задачи, подзадачи, модуля)».
   5.2. Схема алгоритма решения задачи.
   5.3. Код программы на исходном языке программирования с комментариями.
   5.4. Предложения по модификации алгоритмов, кода.

4. Варианты заданий

1 ЗАДАЧА. Написать программу, осуществляющую заданные вычисления с использованием процедур. Вид используемых классов и методов определить самостоятельно.

1.Заданы две матрицы A(4, 4) и B(4, 4). Написать программу суммирования двух векторов X и У, где X - вектор, в котором размещены элементы столбца матрицы A с минимальным средним арифметическим значением, У - то же для матрицы B.

2.Заданы две матрицы A(4, 4) и B(4, 4). Написать программу вычисления вектора *Z*= X+У, где X - строка матрицы A, включающая минимальный элемент ее главной диагонали, У - то же для матрицы B.

3.Заданы две матрицы A(4, 4) и B(3, 3) и два вектора C(4) и D(3). Написать программу вычисления произведений матриц на соответствующие им вектора. Определить минимальное среднее арифметическое значение для полученных векторов.

4.Заданы две матрицы B (4,4) и D(3,3). Написать программу транспонирования каждой из заданных матриц с последующим перемножением транспортированной матрицы на соответствующую ей исходную

5.Заданы две матрицы A(3,3) и B(3,3). Написать программу проверки - является ли произведение этих матриц перестановочным, т.е. проверить равенство AB = BA.

6.Заданы две матрицы C(4,4) и D (3,3). Написать программу определения количества симметричных матриц. Матрица называется симметричной, если транспонированная матрица равна исходной. Для каждой симметричной матрицы вычислить сумму элементов, лежащих вне главной диагонали.

7.Заданы два массива C(6) и 0(10). Для каждого из них осуществить перестановку первого по порядку элемента со вторым отрицательным элементом массива.

8.Заданы три квадратных уравнения $AX^2 + BX + C = 0$; $DX^2 + PX + E = 0$ и $ZX^2 + УX + S = 0$. Написать программу нахождения максимального значения корня среди действительных корней этих уравнений.

9.Заданы два вектора A(0.052; 0.9; 0.15; 0.84; 0.67) и B(0.948; 0.1; 0.33; 0.16; 0.85). Написать программу, позволяющую расположить элементы одного вектора по возрастанию, а другого - по убыванию. Вычислить сумму полученных векторов.

10.Заданы два двумерных массива A (4,4) и B(3,3). Для каждого из них переставить столбцы с максимальным и минимальными элементами.

11.Заданы координаты четырёх точек A, B, C, D на плоскости. Определить наибольший из периметров треугольников ABC, ABD, ACD.

12.Три треугольника заданы координатами своих вершин. Написать программу вычисления площадей треугольников и определения минимальной площади.

13.Заданы два двумерных массива A (4, 4) и B(3,3). Для каждого из них переставить последнюю строку со строкой с максимальным средним арифметическим значением.

14.Заданы две матрицы A(4, 4) и B(4, 4). Написать программу вычисления вектора Z = X - У, где X - столбец матрицы A, включающий минимальный элемент матрицы; У - то же для матрицы B.

15.Заданы три вектора X, У, Z. Написать программу вычисления $S = \sum_{i=1}^{n} C_i \cdot D_i$, где C_i и D_i - компоненты векторов C и D, которые соответственно равны C = X + У;D=X - Z

16.Заданы матрицы C(4, 4) и D(3,3). Определить индексы максимального элемента каждой из матриц среди элементов, расположенных выше главной диагонали.

17.Заданы вектора A(5), B(10), D(15). Дли каждого из них определить максимальный и минимальный элементы и их индексы.

18.Заданы массивы У(4, 4) и Z(8,8). Для каждого из них вычислить среднее арифметическое значение положительных элементов, расположенных выше главной диагонали.

19.Заданы три матрицы размерами 2x2, 3x3, 4x4. Для каждой из матриц определить среднее арифметическое значение положительных элементов главной диагонали.

20.Заданы три одномерных массива разной размерности. Для каждого из массивов определить повторяющиеся элементы.

2 ЗАДАЧА. Познакомиться с основой объектного подхода[2] в языке C#, созданием объектов, классов и механизмом наследования. Построить иерархию классов в соответствии с вариантом задания:

1) Стихотворение, стиль изложения, рифма, проза
2) Служащий, персона, рабочий, инженер
3) Рабочий, кадры, инженер, администрация
4) Деталь, механизм, изделие, узел
5) Организация, страховая компания, нефтегазовая компания, завод
6) Журнал, книга, печатное издание, учебник
7) Тест, экзамен, выпускной экзамен, испытание
8) Место, область, город, мегаполис
9) Игрушка, продукт, товар, молочный продукт
10) Квитанция, накладная, документ, счет
11) Автомобиль, поезд, транспортное средство, экспресс

[2] ПРИНЦИПЫ ОБЪЕКТНОГО ПОДХОДА:

1) **Абстрагирование** – выделение главного, выявление видов абстракций;
2) **Инкапсуляция** – скрываем детали реализации
3) **Иерархия** – разбиение задачи на уровни и постепенное решение
4) **Наследование** – создание новых классов на основе имеющихся
5) **Полиморфизм** – это качество, которое позволяет одному интерфейсу получать доступ к целому классу действий.

12) Двигатель, двигатель внутреннего сгорания, дизель, реактивный двигатель
13) Республика, монархия, королевство, государство
14) Млекопитающее, парнокопытное, птица, животное
15) Товар, велосипед, горный велосипед, самокат
16) Лев, дельфин, птица, синица, животное
17) Музыкант, персона, студент, гитарист
18) Печатное издание, газета, книга, периодика
19) Наземное транспортное средство, корабль, транспортное средство, водное транспортное средство
20) Грузовик, автомобиль, легковое авто, транспорт

5. Контрольные вопросы:

1) Что понимается под термином «класс»?
2) Какие элементы определяются в составе класса?
3) Каково соотношение понятий «класс» и «объект»?
4) Что понимается под термином «члены класса»?
5) Какие члены класса Вам известны?
6) Какие члены класса содержат код?
7) Какие члены класса содержат данные?
8) Перечислите пять разновидностей членов класса специфичных для языка C#.
9) Что понимается под термином «конструктор»?
10) Сколько конструкторов может содержать класс языка C#?
11) Приведите синтаксис описания класса в общем виде.
Проиллюстрируйте его фрагментом программы на языке C#.
12) Какие модификаторы типа доступа Вам известны?
13) В чем заключаются особенности доступа членов класса с модификатором public?
14) В чем заключаются особенности доступа членов класса с модификатором private?
15) В чем заключаются особенности доступа членов класса с модификатором protected?
16) В чем заключаются особенности доступа членов класса с модификатором internal?
17) Какое ключевое слово языка C# используется при создании объекта?
18) Приведите синтаксис создания объекта в общем виде.
Проиллюстрируйте его фрагментом программы на языке C#.
19) В чем состоит назначение конструктора?
20) Каждый ли класс языка C# имеет конструктор?
21) Какие умолчания для конструкторов приняты в языке C#?
22) Каким значением инициализируются по умолчанию значения ссылочного типа?
23) В каком случае конструктор по умолчанию не используется?
24) Приведите синтаксис конструктора класса в общем виде.
Проиллюстрируйте его фрагментом программы на языке C#.

25) Что понимается под термином «деструктор»?
26) В чем состоит назначение деструктора?
27) Приведите синтаксис деструктора класса в общем виде.
Проиллюстрируйте его фрагментом программы на языке C#.
28) Что понимается под термином «наследование»?
29) Что общего имеет дочерний класс с родительским?
30) В чем состоит различие между дочерним и родительским классами?
31) Для чего используется наследование?
32) Опишите синтаксис производного класса. Какие спецификаторы доступа применяются в иерархиях?
33) Как вызвать метод базового класса из производного?
34) Опишите порядок вызова конструкторов базовых классов при работе конструктора производного класса.
35) Какие ключевые слова используется при переопределении методов базового класса в производном?
36) Опишите механизмы раннего и позднего связывания.
37) Опишите процесс вызова виртуального метода. Для чего применяются виртуальные методы?
38) Все ли методы следует описывать как виртуальные?
39) Для чего используются абстрактные классы?