Свойства, деструкторы классов.

Обработка исключительных ситуаций

Лекция №3

Сбор мусора. Деструкторы.

Каждому объекту класса при создании выделяется память в heap.

В C# имеется система сбора мусора, которая автоматически возвращает память для повторного использования. Эта система действует незаметно для программиста, активизируется только по необходимости и точно невозможно узнать, когда происходит сбор мусора.

Деструктор – это специальный метод, который вызывается сборщиком мусора непосредственно перед удалением объекта из памяти.

Деструктор не имеет параметров и не возвращает значение.

Точно неизвестно, когда вызывается деструктор, но все деструкторы будут выполнены перед окончанием программы.

Синтаксис деструктора:

~ имя класса ( )

{ тело конструктора}

тильда

```
class demo_destruct
  {int pole;
   public demo_destruct(int x) // Конструктор
     {pole=x;}
  ~demo_destruct()  // Деструктор
     {Console.WriteLine("Уничтожен объект"+pole);}
  }
  class Program
  {
      static void Main(string[] args)
      {   demo_destruct ob;
      for (int i = 1; i <= 100000; i++)
      {
          Console.WriteLine("Создан объект " + i);
          ob = new demo_destruct(i);

      }
      Console.WriteLine("Все!!!"); Console.ReadKey();
      }
    }
```

Свойства класса

Свойство – это элемент класса, предоставляющий доступ к его полям. Обычно связано с закрытым полем класса.

[атрибуты] [спецификаторы] тип имя_свойства

{

[get код аксессора чтения поля]

[set код аксессора записи поля]

}

Оба аксессора не могут отсутствовать.

Имя свойства можно использовать как обычную переменную в операторах присваивания и выражениях.

При обращении к свойству автоматически вызываются аксессоры чтения и установки.

Обращение к свойству:

имя_объекта.имя_свойства

Аксессор get должен содержать оператор return.

В аксессоре set используется стандартный параметр **value,** который содержит значение устанавливаемое для поля значение.

Например, в классе **Cilindr** из примера выше методы **set_radius** и **get_radius** можно заменить на свойство:

```
public double Radius
        { get { return radius_osnovania; }
          set { radius_osnovania = value; }
```

Обращение к свойству может быть такое:

```
Console.WriteLine("Радиус= {0,5:f3}  Высота= {1,5:f3}",
                    C1.Radius, C1.get_vysota());

C3.Radius = 100; C3.set_vysota(100);
        Console.WriteLine("Новые параметры цилиндра:");
        C3.vyvod();
```

Аксессор set можно дополнить проверкой значения на положительность:

```
public double Radius
    {
      get { return radius_osnovania; }
      set { if (value>0) radius_osnovania = value; }
    }
```

Обработка исключительных ситуаций

Все исключения являются подклассами класса Exception пространства имен System.

Исключения генерирует среда программирования или программист.

Часто используемые исключения пространства имен System:

Исключение	***Значение***
ArrayTypeMismatchException	Тип сохраняемого значения несовместим с типом массива
DivideByZeroException	Попытка деления на ноль
IndexOutOfRangeException	Индекс массива оказался вне диапазона

Outof MemoryException	Обращение к оператору **new** оказалось неудачным из-за недостаточного объема свободной памяти
OverFlowException	Имеет место арифметическое переполнение
FormatException	Попытка передать в метод аргумент неверного формата
InvalidCastException	Ошибка преобразования типа

Свойства класса Exception:

Message **текстовое описание ошибки**

TargetSite **Метод, выбросивший исключение**

Исключения перехватываются и обрабатываются оператором **try.**

try

{контролируемый блок}

catch (тип1 [имя1]) { обработчик исключения1 }

catch (тип2 [имя2]) { обработчик исключения2 }

. . .

catch { обработчик исключения }

finally {блок завершения}

В контролируемый блок включаются операторы, выполнение которых может привести к ошибке.

С try-блоком можно связать не одну, а несколько catch-инструкций. Однако все catch-инструкции должны перехватывать исключения различного типа.

При возникновении ошибки при выполнении операторов контролируемого блока генерируется исключение.

Выполнение текущего блока прекращается, находится обработчик исключения соответствующего типа, которому и передается выполнение.

После выполнения обработчика выполняется блок **finally.**

Блок **finally** будет выполнен после выхода из try/catch-блока, независимо от условий его выполнения.

Если подходящий обработчик не найден, вызывается стандартный обработчик, который обычно выводит сообщение и останавливает работу программы.

Форма обработчика

catch (тип) { обработчик исключения }

используется если важен только тип исключения, а свойства исключения не используются.

Например:

```
 try
    int y=a/b
 catch (DivideByZeroException)
{  Console.WriteLine(" Деление на 0"); }
finelly
  {Console.ReadKey();}
```

Форма обработчика

catch (тип имя) { обработчик исключения }

используется когда имя параметра используется в теле обработчика.

Например:

```
try
   int y=a/b
 catch (DivideByZeroException  f)
{  Console.WriteLine(f.Message+": Деление на 0"); }
```

При попытке деления на 0 выведется сообщение:

Attempted to divide by zero. Деление на 0

Форма обработчика

catch { обработчик исключения }

применяется для перехвата всех исключений, независимо от их типа.

Он может быть только один в операторе try и должен быть помещен после остальных catch-блоков.

```
try
    { int v = Convert.ToInt32(Console.ReadLine()); }
catch { Console.WriteLine("Ошибка!!!"); }
```

В этом примере и в случае ввода очень большого числа и в случае ввода недопустимых в целых константах символов выводится сообщение **"Ошибка!!!"**

Генерирование исключений вручную

Исключение можно сгенерировать вручную, используя инструкцию **throw.**

Формат ее записан таков:

throw [параметр]*;*

Параметр - это <u>объект</u> класса исключений, производного от класса Exception.

объект класса, производного от Exception.

Например:

```
double x;
if (x == 0) throw new DivideByZeroException();
```

```
class Program
    {
        static void D_0(double x)
        { if (x == 0) throw new Exception("Деление на 0"); }
        static void Main(string[] args)
        {
            try
            {   string s = Console.ReadLine();
                double d = Convert.ToDouble(s);
                int v = Convert.ToInt32(Console.ReadLine());
                double dd = Convert.ToDouble(Console.ReadLine());


  D_0(dd);
                  double y; y = d / dd;
                  Console.WriteLine("y=" + y);
                  // D_0(v);
                  double r = 5 / v;
              }
```

```
catch (FormatException)
            { Console.WriteLine("Неверный ввод"); }

        catch (DivideByZeroException f)
             { Console.WriteLine(f.Message+
                    " Деление на 0_для_целых"); }

        catch (Exception gg)
        { Console.WriteLine("Ошибка:   "+gg.Message+" "
                                +gg.TargetSite); }

        finally {Console.ReadKey();}
```

Исключение, перехваченное одной catch-инструкцией, можно перегенерировать, чтобы обеспечить возможность его перехвата другой (внешней) catch-инструкцией.

Чтобы повторно сгенерировать исключение, достаточно использовать ключевое слово throw, не указывая параметра:

throw ;

Например:

```
try
  {
   try
    {
     int v = Convert.ToInt32(Console.ReadLine());

     Console.WriteLine("v=" + v);

     }

    catch (FormatException)
         { Console.WriteLine("Неверный ввод"); throw; }
     }
catch (FormatException)
    { Console.WriteLine("Это очень плохо"); }
```

Один try-блок можно вложить в другой.

Исключение, сгенерированное во внутреннем try-блоке и не перехваченное catch-инструкцией, которая связана с этим try-блоком, передается во внешний try-блок.

Часто внешний try-блок используют для перехвата самых серьезных ошибок, позволяя внутренним try-блокам обрабатывать менее опасные.

Например:

```
try
            {
                try
                {
                  int v = Convert.ToInt32(Console.ReadLine());

                  Console.WriteLine("v=" + v);

                }

                catch (FormatException)
                { Console.WriteLine("Неверный ввод");}
            }
    catch (OverflowException)
  { Console.WriteLine("Слишком большое целое число"); }
```

Использование массивов в C#.

Лекция №4

Массив - это структурированный тип данных, представляющий собой последовательность <u>однотипных</u> элементов, имеющих общее имя и снабженных индексами (порядковыми номерами). Нумерация начинается с нуля.

Массив – это ссылочный тип данных: в стеке размещается ссылка на массив, а сами элементы массива располагаются в динамической области памяти – хипе. Поэтому его нужно создавать с помощью операции **new**.

При создании всем элементам массива присваиваются по умолчанию нулевые значения.

Все массивы в C# построены на основе класса **Array** из пространства имен System**,** а значит, наследуют некоторые его элементы или для них можно использовать его методы.

Одномерный массив

Можно описать одним из следующих способов:

тип[ ] имя_массива;

выражение, тип которого имеет неявное преобразование к int, long, ulong, uint

Например,

double[ ] y, z;

В этом случае память под элементы массивов не выделена.

тип[ ] имя_массива = new тип[размерность];

Например,

int[] a = new int[20], b= new int[100];

В этом случае в памяти создаются массивы из 20 и 100 элементов соответственно, всем элементам присваивается значение 0.

тип[ ] имя_массива = new тип[ ] {список_инициализаторов};

Например,

int[] x = new int[] {2, -5, 0, 9};

В этом случае размерность массива явно не указана и определяется по количеству элементов в списке инициализаторов.

тип[ ] имя_массива = {список_инициализаторов};

new подразумевается.

тип[ ] имя_массива = new тип[размерность] {список_инициализаторов};

Например,

int[] x = new int[4] {2, -5, 0, 9};

В этом случае размерность массива все равно бы определилась, т.е. имеем избыточное описание.

Обращение к элементу массива:

имя массива [индекс]

Например, x[3], MyArray[10]

Ввод массива:

```
Console.WriteLine("Введите количество элементов");
        int n=Convert.ToInt32(Console.ReadLine());
        double[] x = new double[n];
        for (int i = 0; i < n; ++i)
        {
            Console.Write("x[" + i + "]=");
            x[i] = Convert.ToDouble(Console.ReadLine());
        }
```

Оператор цикла **foreach** предназначен для просмотра всех элементов из некоторой группы данных: массива, списка и др.

Синтаксис:

foreach (тип имя_переменной in имя_массива)

тело цикла;

Имя_переменной здает имя локальной переменной, которая будет по очереди принимать все значения из массива, имя которого указано в операторе после слова in.

Ее тип должен соответствовать типу элементов массива.

С помощью оператора **foreach** нельзя изменять значение переменной цикла.

Например, **нельзя написать**

```
foreach (double xt in x)
        {
          xt = Convert.ToDouble(Console.ReadLine());
        }
```

Но это ничуть не улучшает код!!!

Можно так:

```
int j = 0;
        foreach (double xt in x)
        {
          Console.Write("x[" + j + "]=");
          x[j] = Convert.ToDouble(Console.ReadLine());
          ++j;
        }
```

Вывод:

```
for (int i = 0;   i < n;  ++i )
            Console.WriteLine( x[i] );
```

Или с помощью foreach:

```
foreach (double xt in x)
            Console.WriteLine(xt);
```

Можно создать универсальный метод для вывода массива любого типа:

```
static void vyvod(Array z, string zagolovok)
    { Console.WriteLine(zagolovok);
      foreach (object xt in z)
        Console.WriteLine(xt);       }
```

Обращение: **vyvod(x, "Массив x");**

Свойства класса **System.Array:**

Элемент класса	Описание
Length	Количество элементов массива
Rank	Количество размерностей массива

Статические методы:

Clear(x, j,n)	Присваивает **n** элементам массива **x**, начиная с **j**-го, значения по умолчанию. Например: **Array.Clear(x, 1,2);**
BinarySearch(x, xx)	Ищет в отсортированном массиве **x** номер элемента со значением **xx.** Например, **Array.BinarySearch(a, 9)**
Sort(x)	Упорядочивает массив **x** в порядке возрастания значений элементов

Sort(x, j, n)	Упорядочивает часть массива **x** из n элементов, начиная с j-го
Reverse(x)	Изменяет порядок следования элементов массива x на обратный.
Reverse(x, j, n)	Изменяет порядок следования n элементов массива x на обратный, начиная с j-го элемента.

Copy(x,z,n)	Копирует n элементов массива x в массив z (n должно быть не больше размерности z и x)
Copy(x,j,z,k,n)	Копирует n элементов массива x, начиная с j-го, в массив z с k-й позиции

IndexOf(x, xx)	Ищет в массиве **x** номер первого элемента со значением **xx.**
LastIndexOf(x, xx)	Ищет в массиве **x** номер последнего элемента со значением **xx.**

Нестатические методы:

CopyTo(x,j)	Копирование всех элементов массива, для которого вызван метод, в массив x, начиная с j-го элемента. Например, **x.CopyTo(z, 1);**
GetValue(j)	Получение значения элемента массива с номером j (**a.GetValue(3)**)
SetValue(xx, j)	Установка значения xx для j-го элемента массива **z.SetValue(8,1) ;**

Пример. Вычислить значение функции

$$z = 3\cos^2 3a - 4\sin b - c^{2.1}$$

, где a, b и c – количество положительных элементов в массивах A, B и C соответственно.

Для решения задачи создать класс «Массив», содержащий закрытое поле для хранения данных, методы ввода и вывода элементов массива, свойство (только для чтения) – «количество положительных элементов».

Массивы A и B вводить с клавиатуры, массив C сформировать, скопировав сначала все элементы массива A, а затем первые три элемента массива B.

```
class Massiv
    { double[ ] a;
      public Massiv(int  n) { a = new double[n]; }


      public double[ ] a1
            {
                get {return a;}
                 set{ a=value;}
              }
```

```
public int kolich_polog
        {
            get { int k = 0;
                    foreach (double x in a)
                        {if  (x>0)   ++k;}
                      return k;
                  }
        }
```

```
public void vyvod( string zagolovok)
        { Console.WriteLine(zagolovok);
         foreach (double x in a)
              Console.WriteLine(x);
        }
```

```
public void vvod( string name )
    {
        Console.WriteLine("Введите элементы массива " + name);
        for (int i = 0; i < a.Length; i++)
        {
          a[i] = Convert.ToDouble(Console.ReadLine());
        }
    }


}
```

```
static void Main(string[] args)
    {
     Console.WriteLine("Введите размер  массива A");
      Massiv A = new Massiv(Convert.ToInt32(Console.ReadLine()));
      A.vvod("A");


         Console.WriteLine("Введите размер  массива B");
          Massiv B = new Massiv(Convert.ToInt32(Console.ReadLine()));
          B.vvod("B");
```

```
Massiv C = new Massiv(A.a1.Length+3);
A.a1.CopyTo(C.a1,0);
Array.Copy(B.a1, 0, C.a1, A.a1.Length, 3);
```

```
A.vyvod("Массив A"); B.vyvod("Массив B"); C.vyvod("Массив C");
Console.WriteLine("Количество положительных элементов в A=" + A.kolich_polog);
Console.WriteLine("Количество положительных элементов в B=" + B.kolich_polog);
Console.WriteLine("Количество положительных элементов в C=" + C.kolich_polog);
```

```
double z = 3 * Math.Pow(Math.Cos(3 * A.kolich_polog), 2) - 4 *
Math.Sin(B.kolich_polog) - Math.Pow(C.kolich_polog, 2.1);
        Console.WriteLine("Z={0,6:f3}", z);
        Console.ReadKey();
    }
  }
}
```

Пример. Задать параметры N прямоугольников. Определить количество прямоугольников, площадь которых превышает заданное число. Получить список прямоугольников, которые являются квадратами.

```
class Pramoug
  { double vys, shir;
   public Pramoug(double v, double sh)
   {vys = v; shir = sh;}
   public double Vys
   {
      get { return vys; }
      set { vys = value; }
   }
```

```
public double Shir
    {
        get { return shir; }
        set { shir = value; }
    }
    public double Plosh
    {
        get { return vys*shir; }


    }


}
```

```
class Program
  {
     static void Main(string[] args)
     {    Console.WriteLine("Сколько прямоугольников");
    Pramoug[] pr=new Pramoug[Convert.ToInt32(Console.ReadLine())];
        for (int i = 0; i < pr.Length; ++i)
        {
Console.WriteLine("задайте параметры " + (i+1) + "-го
                          прямоугольника");
  pr[i] = new Pramoug(Convert.ToDouble(Console.ReadLine()),
                              Convert.ToDouble(Console.ReadLine()));
        }
```

```
Console.WriteLine("Задайте пороговую площадь");
double pl=Convert.ToDouble(Console.ReadLine());
    int k = 0;
    foreach (Pramoug p in pr)
        { if (p.Plosh  > pl)   ++k; }

Console.WriteLine("У "+k+
                    " прямоугольников площадь превышает "+pl);
```

```
Console.WriteLine("Квадраты:");
    foreach (Pramoug p in pr)
    { if (Math.Abs(p.Shir-p.Vys)<0.1e-9)
        Console.WriteLine(Array.IndexOf(pr,p)+"-й прямоугольник"); ;
     }
     Console.ReadKey();
        }
    }
}
```

Многомерные массивы

Многомерным называется массив, который характеризуется двумя или более измерениями, а доступ к отдельному элементу осуществляется посредством двух или более индексов.

Существуют два типа многомерных массивов: ***прямоугольные*** и ***ступенчатые (разреженные, рваные, зубчатые).***

Вот как объявляется многомерный прямоугольный массив:

тип[, … ,] имя = new тип[размер_1, ..., размер_N] ;

Например:

int [, ,] y = new int [4 , 3, 3];

Так создается трехмерный целочисленный массив размером **4x3x3.**

Простейший многомерный массив — **двумерный**, в котором позиция любого элемента определяется двумя индексами.

Двумерный массив можно описать одним из следующих способов:

тип[,] имя_массива;

Например,

double[,] y, z;

В этом случае память под элементы массивов не выделена.

тип[,] имя_массива = new тип[разм_1, разм_2];

Например,

int[,] a = new int[5,5], b= new int[10,4];

В этом случае в памяти создаются массивы из 25 и 40 элементов соответственно, всем элементам присваивается значение 0.

тип[,] имя_массива = new тип[,] {список_инициализаторов};

Например,

```
int[,]  x = new int[,] {{2, -5,  0, 9},
                        {3, 2, -5, 5},
                        {2, 4, 6, -1}};
```

значения сгруппированы в фигурных скобках по строкам

В этом случае размерность массива явно не указана и определяется по количеству элементов в списке инициализаторов.

тип[,] имя_массива = {список_инициализаторов};

new подразумевается.

тип[ ] имя_массива = new тип[разм1, разм2] {список_инициализаторов};

Например,

```
int[ , ] x = new int[2,2] {{2, -5}, { 0, 9}};
```

В этом случае размерность массива все равно бы определилась, т.е. имеем избыточное описание.

Обращение к элементу матрицы:

имя массива [индекс1, индекс2]

Например, x[3,4], MyArray[1,0]

Ввод матрицы:

```
Console.WriteLine("Введите кол-во строк:");
int n = int.Parse(Console.ReadLine());
Console.WriteLine("Введите кол-во столбцов:");
int m = int.Parse(Console.ReadLine());
double[,] x = new double[n, m];

for (int i=0;i<n;++i)
  {Console.WriteLine(
      "Введите элементы "+i+"-й строки:");
   for (int j = 0; j < m; ++j)
    x[i, j] = double.Parse(Console.ReadLine());
   }
```

Вывод матрицы:

```
for (int i = 0; i < n; ++i)
      {
       for (int j = 0; j < m; ++j)
        Console.Write("{0,5:f2}",x[i,j]);
       Console.WriteLine();
      }
```

Применение оператора **foreach** для просмотра прямоугольных массивов.

Повторение оператора **foreach** начинается с элемента, все индексы которого равны нулю, и повторяется через все возможные комбинации индексов с приращением крайнего правого индекса первым.

Когда правый индекс достигает верхне1 границы, он становится равным нулю, а индекс слева от него увеличивается на единицу.

Например, пусть требуется найти максимальный элемент матрицы.

```
double max = x[0, 0];
foreach   (double xt in x)  if   (xt > max)   max = xt;
            Console.WriteLine("Максимум: "+max);
```

Нестатические м*етоды класса Array для работы с двумерными массивами:*

GetLength(n *)*	Возвращает длину заданной размерности. Например, если размерность матрицы A 3×5, то **A. GetLength(1)** будет равно 5.
GetValue(i ,j)	Возвращает значение элемента вызывающего массива с индексами [i, j].
SetValue(xx, i, j)	Устанавливает в вызывающем массиве элемент с индексами [i,j] равным значению xx

Метод для вывода массива:

```
static void vyvod_matr( Array v,string zagolovok)
   { Console.WriteLine(zagolovok);
     for (int i=0;i<v.GetLength(0);++i)
      {for (int j = 0; j < v.GetLength(1); ++j)
         Console.Write("{0,5:f2}",v.GetValue(i,j));
        Console.WriteLine();
       }
    }
```

Пример. Если максимальный элемент матрицы A больше максимального элемента матрицы B, увеличить все элементы матрицы A на значение максимального элемента, в противном случае уменьшить в два раза положительные элементы матрицы B.

```
class Matrica
    {
        int[,] x;
 public Matrica()
        {
            Console.WriteLine("Введите кол-во строк:");
            int n = int.Parse(Console.ReadLine());
            Console.WriteLine("Введите кол-во столбцов:");
            int m = int.Parse(Console.ReadLine());
            x = new int[n, m];
        }
```

```
public Matrica(int n, int m) { x = new int[n, m]; }

 public int[,] X
        {
            get { return x; }
            set { x = value; }
        }

 public int max
        { get { int mx = x[0,0];
                foreach(int xt in x) if (xt > mx) mx = xt;
                return mx;}
        }
```

```
public void vvod(string name)
    {
        Console.WriteLine("Введите матрицу " + name);
        for (int i = 0; i < x.GetLength(0); ++i)
        {
                Console.WriteLine("Введите элементы " + i + "-й
строки:");
            for (int j = 0; j < x.GetLength(1); ++j)
                x[i, j] = int.Parse(Console.ReadLine());
        }
    }
```

```
public void vyvod_matr(string zagolovok)
    {
        Console.WriteLine(zagolovok);
        for (int i = 0; i < x.GetLength(0); ++i)
        {
            for (int j = 0; j < x.GetLength(1); ++j)
                Console.Write(" {0,7:f2}", x[i, j]);
            Console.WriteLine();
        }
    }
```

```
class Program
    {       static void Main(string[] args)
        {

 Console.BackgroundColor = ConsoleColor.Cyan; Console.Clear();
            Console.ForegroundColor = ConsoleColor.Black;

Matrica A = new Matrica(); A.vvod("A");
 Matrica B = new Matrica(3, 4); B.vvod("B");

            A.vyvod_matr("Исходная матрица A");
             B.vyvod_matr("Исходная матрица B");
```

```
int max_A = A.max;

if (A.max > B.max)
        {
            for (int i = 0; i < A.X.GetLength(0); ++i)
            {
                for (int j = 0; j < A.X.GetLength(1); ++j) A.X[i, j] = A.X[i, j] + max_A;
            }
            A.vyvod_matr("Новая A");
        }
```

```
else
        {
            for (int i = 0; i < B.X.GetLength(0); ++i)
            {
                for (int j = 0; j < B.X.GetLength(1); ++j)
                    if (B.X[i, j] > 0) B.X[i, j] = B.X[i, j] / 2;
            }
            B.vyvod_matr("Новая B");
        }
Console.ReadKey();
    }
  }
```

Ступенчатые массивы – это массивы, в которых количество элементов в разных строках может быть различным.

Поэтому такой массив можно использовать для создания таблицы со строками разной длины.

Представляет собой массив массивов, в памяти хранится в виде нескольких внутренних массивов, каждый из которых имеет свою длину, а для хранения ссылки на каждый из них выделяется отдельная область памяти.

Описание:

тип[ ] [ ] имя = new тип[размер] [ ];

Например, int[ ] [ ] z = new[10] [ ];

Здесь **размер** означает количество строк в массиве.

Для самих строк память выделяется индивидуально. Под каждый из внутренних массивов память требуется выделять явным образом.

Например:

```
int [ ] [ ]  x = new int [ 3 ] [ ] ;
             x[0] = new int [ 4 ] ;
             x[1] = new  int[ 3 ] ;
             x[2] = new int [ 5 ] ;
```

В данном случае x.Lenght равно 3,

x[0].Lenght равно **4**, x[1].Lenght равно **3**, x[2].Lenght равно **5**.

Другой способ:

тип[ ] [ ] имя = {создание_массива1, создание_массива2,…,

создание_массиваN};

Например,

int [ ] [ ] x = { new int [4] , new int[3], new int [5] };

Доступ к элементу осуществляется посредством задания каждого индекса внутри своих квадратных скобок.

имя[индекс1] [индекс2]

Например, **x[2] [9] или a[i] [j]**

Пример. Определить средний балл в группах студентов, вывести списки двоечников в каждой группе, назначить студентам стипендию, которой они достойны.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication1
{
```

```
class Student
    {
        int[ ] ocenki;
        string fam, gruppa;
        static double mrot;


public Student(string fam, string gruppa, int n)
 { this.fam = fam; this.gruppa = gruppa;
   ocenki = new int[n]; }
```

```
public string Fam
        {
            set { fam = value; }
            get { return fam; }
        }

public string Gruppa
        {
            set { gruppa = value; }
            get { return gruppa; }
        }
```

```
 static Student()
        { mrot = 200000; }


public void vvod_oc()
    {
 Console.WriteLine("Введите оценки студента "
+ fam + " за сессию");
for (int i = 0; i < ocenki.Length;++i )
 {ocenki[i]= int.Parse(Console.ReadLine()); }
     }
```

```
public double Sr_b
 {
    get { double S = 0;
        foreach (int x in ocenki) S = S + x;
        return S / ocenki.Length;}
  }
```

```
public double stip
 {
  get { if (Dvoechnik) return 0;
         else return
           (Sr_b > 8) ? (2 * mrot) :
        (Sr_b > 6 ? 1.8 * mrot : 1.25 * mrot);
        }
  }
```

```
public bool Dvoechnik
  {
   get
    {if (Array.IndexOf(ocenki, 2)>=0)
                 return true;
     else
                 return false;
     }
  }

}
```

```
class Program
  {
    static void Main(string[ ] args)
     {   Console.WriteLine("Сколько групп?");
         int m=int.Parse(Console.ReadLine());
         Student[ ][ ] fakultet = new Student[m][ ];
         for (int i = 0; i < m; ++i)
           {
             Console.WriteLine("Какая группа?");
              string grp = Console.ReadLine();
```

```
Console.WriteLine("Сколько в группе студентов ?");
 int ks = Convert.ToInt32(Console.ReadLine());
 fakultet[i]=new Student[ks];
  for (int j = 0; j < ks; ++j)
  {   Console.WriteLine("Фамилия студента ? ");
      fakultet[i][j] = new Student(Console.ReadLine( ), grp, 4);
      fakultet[i][j].vvod_oc( );
   }
}
```

```
for (int i = 0; i < m; ++i)
  {
    double Sb = 0;
   foreach (Student xs in fakultet[i])
            { Sb = Sb + xs.Sr_b; }
Console.WriteLine(fakultet[i][0].Gruppa + "   "+Sb/fakultet[i].Length);
   }
```

```
for (int i = 0; i < m; ++i)
        {

Console.WriteLine("\n Двоечники группы " + fakultet[i][0].Gruppa);
          foreach (Student xs in fakultet[i])
            {
                if (xs.Dvoechnik)
                  Console.WriteLine(xs.Fam);
            }
          }
```

```
foreach (Student[ ] xss in fakultet)
 {
  Console.WriteLine(
                "\nСтипендия студентов группы "+ xss[0].Gruppa);

   foreach (Student xs in xss)
        {
                Console.WriteLine(xs.Fam+"  "+xs.stip);
         }
 }        Console.ReadKey();
       }
    }
 }
```