Программирование на C#

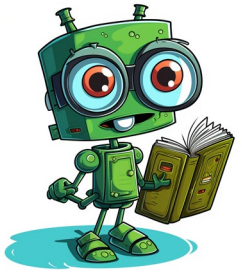
Семинар №3

Модуль №2

Тема:

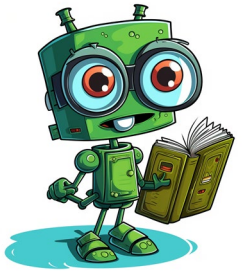
Классы. Массивы объектов
Наследование.

Полезные материалы к семинару



- Console.ForegroundColor (https://learn.microsoft.com/ru-ru/dotnet/api/system.console.foregroundcolor?view=net-5.0)
- Наследование (https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/tutorials/inheritance)

Задания преподавателя к семинару



- Разобраться с массивами объектов
- Познакомиться с наследованием

Demo 01. Правильный многоугольник.

Класс **правильных** многоугольников (**Polygon**):

- Конструктор с умалчиваемыми значениями аргументов играет роль конструктора по умолчанию.
- Поля класса – число сторон (целое) и радиус вписанной окружности (вещественный).
- Свойства – периметр и площадь многоугольника.
- Общедоступный метод **PolygonData()** формирует и возвращает строку со значениями полей и свойств объекта.

В основной программе определить одну ссылку с типом класса.

Создать объект, используя конструктор по умолчанию, затем вводить характеристики многоугольника и выводить сведения о них.

Demo 01. Правильный многоугольник.

```
1.  // Класс многоугольник.
2.  public class Polygon
3.  {
4.      // Число сторон.
5.      int numb;
6.      // Радиус вписанной окружности.
7.      double radius;
8.      // Конструктор по умолчанию.
9.      public Polygon(int n = 3, double r = 1)
10.     {
11.         numb = n;
12.         radius = r;
13.     }

14.     // Периметр многоугольника - свойство.
15.     public double Perimeter
16.     {
17.         get
18.         {   // Аксессор свойства.
19.             // ToDo 01: реализовать свойство
    подсчета периметра многоугольника
20.         }
21.     }
23.     // Площадь многоугольника - свойство.
24.     public double Area
25.     {
26.         get
27.         {   // Аксессор свойства.
    28.  //ToDo 02: реализовать свойство
    подсчета площади многоугольника
29.         }
30.     }
31.          // Данные о многоугольнике -
метод.
32.     public string PolygonData()
33.     {
34.         string res = string.Format("N={0};
35.         R={1}; P={2:F3}; S={3:F3}",
36.         numb, radius, Perimeter, Area);
37.         return res;
38.     }
39. }
```

Demo 01. Правильный многоугольник.

```
1. public static void Main(string[] args)
2.     {

3.         Polygon polygon = new Polygon();
4.         Console.WriteLine("По умолчанию создан многоугольник: ");
5.         Console.WriteLine(polygon.PolygonData());

6.         double rad;
7.         int number;
8.         do
9.         {
10.            do Console.Write("Введите число сторон: ");
11.            while (!int.TryParse(Console.ReadLine(), out number) | number < 3);

12.            do Console.Write("Введите радиус: ");
13.            while (!double.TryParse(Console.ReadLine(), out rad) | rad < 0);

14.            polygon = new Polygon(number, rad);
15.            Console.WriteLine("Сведения о многоугольнике:");
16.            Console.WriteLine(polygon.PolygonData());

17.            Console.WriteLine("Для выхода нажмите клавишу ESC");
18.        } while (Console.ReadKey(true).Key != ConsoleKey.Escape);

19.    }
```

Self 01. Массив Правильных Многоугольников.

Задание выполняется на основе программы Demo 01.

Создайте в основной программе массив объектов типа **Polygon**, количество объектов получите от пользователя. Данные по каждому объекту вводит пользователь.

Программа определяет площади всех многоугольников и выводит данные о всех объектах. Площадь объекта с минимальной площадью выводится **зелёным** цветом, площадь объекта с максимальной площадью выводится **красным** цветом.

Self 02*. Ввод Массива Правильных Многоугольников.

Модифицировать программу Self 01, убрав необходимость ввода количества объектов. Ввод данных продолжается для тех пор, пока не будет введён объект с нулевыми характеристиками. Информация обо всех введённых объектах выводится на экран после добавления каждого нового объекта.

Demo 02. Точка.

Создать класс, описывающий точку в трехмерном пространстве (**Point**).

- Координаты точки – вещественные поля класса.

- Описать конструктор без параметров и конструктор с параметрами – координаты точки.

- Предусмотреть свойства доступа к полям класса.

- Описать метод, вычисляющий расстояние от точки (объекта класса) до точки, координаты которой переданы в параметрах метода.

В основной программе создать три объекта класса и вывести расстояние от этих точек до начала координат.

Demo 02. Точка.

```
1. // Класс точка.
2. class Point
3. {
4.     // Свойства.
5.     public double X { get; set; }
6.     public double Y { get; set; }

7.     // Конструктор без параметров.
8.     public Point()
9.     {
10.        X = 0;
11.        Y = 0;
12.    }

13.    // Конструктор с параметрами.
14.    public Point(double x, double y)
15.    {
16.        X = x;
17.        Y = y;
18.    }

19.    // Метод, вычисляющий расстояние от точки.
20.    public double Distance(Point point)
21.    {
22.        return Math.Sqrt((X - point.X) * (X - point.X) +
23.            (Y - point.Y) * (Y - point.Y));
24.    }
25. }
```

Demo 02. Точка.

```
1. public static void Main(string[] args)
2.     {
3.         Point zero = new Point(0, 0);
4.         Point first = new Point(3, 4);
5.         Point second = new Point(1.1, 2.2);
6.         Point third = new Point(0, 5);
7.         Point[] points = new Point[3] { first, second, third };

8.         for (int i = 0; i < 3; i++)
9.         {
10.            Console.WriteLine($"Расстояние от точки с координатами " +
11.                $"{points[i].X} и {points[i].Y} до начала координат равно
   {points[i].Distance(zero)}.");        }
12.    }
```

ToDo 03. Точка.

Изменить основную программу в задаче Demo 02 следующим образом:

Создать массив из **n** объектов типа **Point**, где **n** – случайное число из интервала **[5, 15]**. Координаты точек заполнить случайными числами в интервале **[-10, 10]**. Вывести на экран информацию обо всех объектах массива, а также расстояние от каждой точки до начала координат.

Self 03. Треугольник.

Создать класс, описывающий треугольник на плоскости (**Triangle**):

– Треугольник задаётся координатами вершин (поля класса типа **Point**. **Point** – точка на плоскости, по аналогии с задачей Demo02).

– Описать конструктор без параметров и два конструктора с параметрами – координаты вершин или точки на плоскости.

– Предусмотреть свойства доступа полям класса.

– Описать свойства «периметр» и «площадь» треугольника.

В основной программе создать массив из **N** объектов типа **Triangle**, где **N** – случайное число из интервала **[5, 15]**. Координаты точек треугольника заполнить случайными числами в интервале **[-10, 10]**.

– Вывести на экран информацию обо всех объектах массива.

– Отсортировать массив треугольников по убыванию их площади.

Инкапсуляция данных в классах и цикл повтора решения обязательны.
Обязательно выводите промежуточные значения на экран.

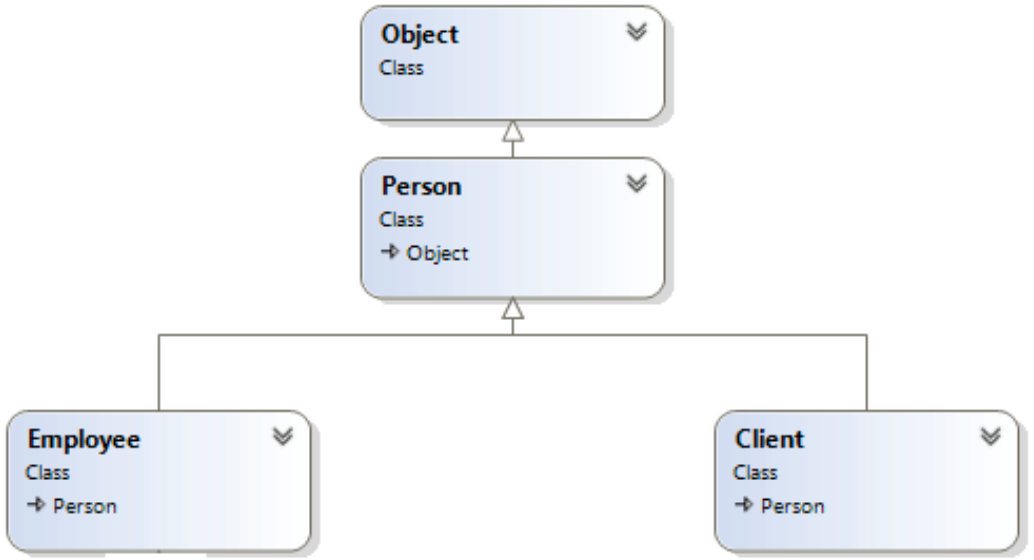
Self 04*. Многоугольник.

Изменить программу Self 03, обобщив класс до многоугольников.

– Реализовать в классе конструктор, принимающий неопределённое количество параметров – координаты углов многоугольника (использовать ключевое слово **params**).

– В конструкторе сделать проверку на существование многоугольника. Если его не существует – бросать исключение типа InvalidArgumentException.

Demo 03. Наследование.




```
class Person { }
    class Employee : Person { }
    class Client : Person { }
```

Demo 03. Наследование.

```
1. class Person
2. {
3.     private string _name = "";

4.     public string Name
5.     {
6.         get { return _name; }
7.         set { _name = value; }
8.     }

9.     public void Print()
10.     {
11.         Console.WriteLine($"Этого человека зовут
  {Name}");
12.     }
13.}
```

Demo 03. Наследование.

```
1.class Employee : Person
2.{
      // ToDo 04: реализовать метод PrintName().
1.}

2.sealed class Client : Person
3.{

4.}
```

Demo 03. Наследование.

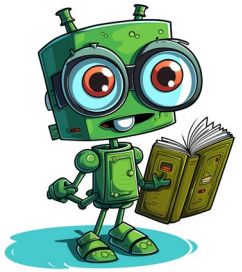
```
1. public static void Main(string[] args)
2.     {
3.         Person person = new Person { Name = "Tom" };
4.         person.Print();

5.         person = new Employee { Name = "Sam" };
6.         person.Print();

7.         Employee employee = new Employee { Name =
   "Bob" };
8.         employee.PrintName();

9.     }
```

Наследование



- По умолчанию все классы наследуются от базового класса **Object**, даже если мы явным образом не устанавливаем наследование. Поэтому выше определенные классы **Person**, **Employee** и **Client** кроме своих собственных методов, также будут иметь и методы класса **Object**: ToString(), Equals(), GetHashCode() и GetType().
- Не поддерживается множественное наследование, класс может наследоваться только от одного класса.
- При создании производного класса надо учитывать тип доступа к базовому классу - тип доступа к производному классу должен быть таким же, как и у базового класса, или более строгим. То есть, если базовый класс у нас имеет тип доступа **internal**, то производный класс может иметь тип доступа **internal** или **private**, но не **public**.
- Если класс объявлен с модификатором **sealed**, то от этого класса нельзя наследовать и создавать производные классы.
- Нельзя унаследовать класс от статического класса.

Demo 04. Конструкторы.

```
1. class A
2. {
3.     public A(int a)
4.     {
5.         Console.WriteLine("A");
6.     }
7. }

8. class B : A
9. {
10.    public B() : base(0)
11.    {
12.        Console.WriteLine("BBB");
13.    }

14.    public B(int b) : base(b)
15.    {
16.        Console.WriteLine("B");
17.    }
18.}
```

```
1. class C : B
2. {
3.     public C() : base(0)
4.     {
5.         Console.WriteLine("CCC");
6.     }

7.     public C(int c) : this()
8.     {
9.         Console.WriteLine("C");
10.    }
11.}
```

ToDo 05. Конструкторы.

```
1. public static void Main(string[] args)
2. {
3.     //A aa = new A(0);

4.     //A ab = new B();

5.     //A ac = new C(0);

6.     //B bb = new B();
7.     //B bc = new C();

8.     //C cc = new C();
9. }
```

Раскоментируйте по очереди каждую строку.

Какие конструкторы будут вызваны при создании каждого объекта?

Demo 05. Транспортное средство.

```
1. // Класс Транспортное средство.
2. public class Transport
3. {
4.     // Год выпуска, вес, цвет.
5.     public int Year { get; set; }
6.     public int Weight { get; set; }
7.     public string Color { get; set; }

8.     // Конструктор без параметров.
9.     protected Transport() { }

10.    // Конструктор с параметрами.
11.    protected Transport(int year, int weight, string
   color)
12.    {
13.        Year = year;
14.        Weight = weight;
15.        Color = color;
16.    }

17.    // Информация о транспортном средстве.
18.    public void Info()
19.    {
20.        Console.WriteLine("Transport");
21.        Console.WriteLine($"Year: {Year}\n" +
22.                          $"Weight: {Weight}\n" +
23.                          $"Color: {Color}");
24.    }
25.}
```

Demo 05. Транспортное средство.

```
1.  // Класс машина.
2.  public class Car : Transport
3.  {
4.      // Скорость.
5.      public double Speed { get; set; }

6.      // ToDo06: Реализовать конструктор с
    параметрами, вызывающий базовый конструктор.

7.      // Информация о машине.
8.      public void Info()
9.      {
10.         base.Info();
11.         Console.WriteLine($"Speed:
    {Speed:0.00}");
12.     }
13. }
```

```
1. // Класс грузовая машина.
2. public class Truck : Car
3. {
4.     // Длина машины.
5.     public double BodyLength { get; set;
   }

6.     // ToDo07: Реализовать конструктор с
   параметрами, вызывающий базовый
   конструктор.

7.     // Информация о грузовой машине.
8.     public void Info()
9.     {
10.        base.Info();
11.        Console.WriteLine($"BodyLength:
   {BodyLength:0.00}\n");
12.    }
13.}
```

Demo 05. Транспортное средство.

```
1.  public static void Main(string[] args)
2.      {
3.          Random rand = new Random();
4.          string[] colors = new string[] { "Blue", "Black", "Red",
    "Green", "Yellow" };

5.          Truck[] arr = new Truck[3];

6.          for (int i = 0; i < arr.Length; i++)
7.          {
8.              string colorTruck = colors[rand.Next(0, colors.Length)];
9.              arr[i] = new Truck(rand.Next(20, 50), rand.Next(5000),
    colorTruck, rand.NextDouble() * 10 + 120, rand.NextDouble() * 2 + 3);
10.         }

11.         foreach (var transport in arr)
12.         {
13.             transport.Info();
14.         }
15.     }
```

Self 05. Студент.

Реализовать класс, представляющий сведения о человеке **Person**.

Реализовать свойства:

- Ф.И.О.(string FullName)
- Дата рождения (DateTime BirthDate),
- Пол (bool IsMale).

Реализовать метод для вывода информации о человеке void ShowInfo().

Реализовать класс, представляющий сведения о студенте **Student** (наследуется от **Person**).
Реализовать свойства:

- Название ВУЗа (string Institute)
- Специальность (string Speciality).

Self 06. Работник.

Реализовать класс, представляющий сведения о сотруднике фирмы **Employee** (наследуется от **Person**).

Реализовать свойства:

- Название компании (string CompanyName)
- Должность (string Post)
- График (string Schedule)
- Оклад (decimal Salary).

В основной программе решить задачи:

- Создать объекты всех трех типов и вызвать ShowInfo(), чтобы показать всю доступную информацию.

- Создать массив **Person**[] arr и присвоить его членам объекты всех трех типов. Продемонстрировать работу метода ShowInfo() на массиве.