

Программирование

Тема 4.2 C#. Введение в классы и объекты. Передача параметров в методы

C#. Классы и объекты

● Описание класса:

```
[ Модификатор доступа ] class ИмяКласса [ : System.Object]
{
    // члены класса – поля и методы
    Модификатор доступа Тип имяПоля [ = значение ];
    Модификатор доступа Тип имяМетода (Параметры) {
        // тело метода }
    }
}
```

● Модификаторы доступа и доступ для членов класса:

- **public** – общий (неограниченный) доступ;
- **protected** – доступ ограничен в пределах данного класса и классов, производных от данного;
- **internal** - доступ ограничен сборкой, в которой находится данный класс;
- **protected internal** - доступ ограничен в пределах данного класса и классов, производных от данного, или - доступ ограничен сборкой, в которой находится данный класс;
- **private** - доступ ограничен в пределах данного класса;
- Доступом по умолчанию для членов класса является **private**

C#. Классы и объекты

- Класс – это конструкция языка программирования, которая объединяет в себе переменные и функции (методы), которая может определять новый тип данных, который можно использовать для создания объектов этого типа.
- В C# как и в Java объявление класса совмещено с его определением, то есть методы класса нужно реализовывать (писать код тела) сразу же при их описании. Поля класса могут инициализироваться при объявлении.
- Модификаторы доступа - это ключевые слова, которые определяют доступность класса или его членов.
- В языке C# предусмотрены следующие модификаторы доступа: **public**, **protected**, **internal**, **protected internal**, **private**.
- Для членов класса (полей и методов) разрешено использование всех указанных выше модификаторов.
- Модификатор доступа должен появляться перед каждым отдельным полем или методом (иначе данный элемент будет иметь уровень доступа private).
- Поля и методы класса, помеченные как public, доступны через объект по имени из классов текущей сборки и из классов других сборок (если класс имеет уровень доступа public).
- Поля и методы класса, помеченные как protected, доступны по имени только из методов своего класса и классов потомков.
- Поля и методы класса, помеченные как protected internal, доступны по имени только из методов своего класса и классов потомков, а также доступны через объект по имени для любого класса текущей сборки.
- Private-поля и private методы доступны по имени только из методов своего класса.
- Поля и методы класса, помеченные как internal, доступны по имени только из методов своего класса и классов потомков текущей сборки, а также доступны через объект по имени для любого класса текущей сборки.
- Доступ для классов верхнего уровня (классов, не вложенных в другие классы) ограничивается модификаторами internal и public (по умолчанию – internal).

- Создание объектов:

ИмяКласса имяОбъекта; // объявление ссылки на объект
имяОбъекта = new ИмяКласса([параметры конструктора]);

- Доступ к полям и методам объекта:

имяОбъекта.имяПоля = значение;
ИмяОбъекта.имяМетода(параметры);

- Варианты конструкторов:

- Конструкторы с параметрами;
- Конструкторы без параметров (конструкторы по умолчанию).

- Специальная ссылка **this**

C#. Классы и объекты

- Объект в C# как и в Java – это экземпляр класса. Класс как и в Java так и в C# является ссылочным типом данных, поэтому механизм создания объектов одинаков:
 1. Создается ссылочная переменная в стеке для хранения адреса будущего объекта;
 2. В динамической памяти (куче) выделяется пространство для размещения объекта со всеми его полями;
 3. Поля объекта инициализируются значениями по умолчанию;
 4. Выполняется явная инициализация полей объекта, если она была задана программистом;
 5. Выполняется конструктор;
 6. Ссылка на созданный объект (его адрес) записывается в соответствующую ссылочную переменную.
- В C# доступ к элементам объекта вне класса, которому принадлежит объект, такой же как и в Java - используется операция «точка» после переменной-ссылки на объект, если конечно этот доступ разрешен модификаторами доступа. Если элементы объекта имеют модификатор доступа private, то доступ к ним осуществляется через public-методы.
- Идеология конструкторов в C# мало чем отличается от конструкторов в C++ и Java. Конструктор – это метод, имя которого совпадает с именем класса и который вызывается автоматически при создании объекта. Типы конструкторов в C# те же что и в Java: конструкторы с параметрами и конструкторы без параметров (конструкторы по умолчанию).
- Ссылка this – это скрытый параметр метода, который хранит ссылку на текущий объект, который вызывает метод.
- Как и в Java в C# уничтожение неиспользуемых объектов осуществляется автоматически «сборщиком мусора», специальным механизмом CLR. В C# можно создать метод вида ~имя_класса(), который полностью аналогичен методу finalize() в Java, то есть является завершителем, который вызывается механизмом CLR перед уничтожением объекта. В C# метод вида ~имя_класса() также называют деструктором.

● Пример 1:

```
using System;
namespace Primer1
{
    class Dog
    {
        private string name; //кличка
        private int age; //возраст
        public string getName(){return name;}
        public int getAge() { return age; }
        public void setName(string newName) { name = newName; }
        public void setAge(int newAge) { age = newAge; }
        public Dog() { name = "NoName"; age = 1; }
        public Dog(string name, int age)
        { this.name = name; this.age = age; }
        public void voice()
        { Console.Write("{0}: ", name);
          for (int i = 1; i <= age; i++)
              Console.Write("гав ");
          Console.WriteLine();
        }
    }
}
```

C#. Классы и объекты

● Пример 1:

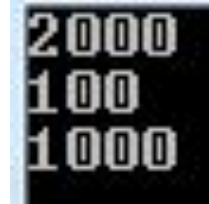
```
class Program
{ static void Main(string[] args)
  { Dog dog1 = new Dog();
    dog1.voice();
    Dog dog2 = new Dog("Тузик", 3);
    dog2.voice();
    dog1.setName("Пушок");
    dog1.setAge(4);
    dog1.voice();
    Console.ReadKey();
  }
}
```

```
NoName : гав
Тузик : гав гав гав
Пушок : гав гав гав гав
```

C#. Статические поля и методы

● Пример

```
using System;
namespace Primer2
{
    class SomeClass
    { // объявление константы подразумевает слово static !!!
        const int T = 2000;
        // SomeClass() { }
        public static int i = 100;
        public static int GetI() { return i; }
        public int GetT() { return T; }
    }
    class Test
    { static void Main()
        {
            SomeClass s = new SomeClass();
            Console.WriteLine(s.GetT());
            Console.WriteLine(SomeClass.GetI());
            SomeClass.i = 1000;
            Console.WriteLine(SomeClass.GetI());
            Console.ReadKey();
        }
    }
}
```



```
2000
100
1000
```


С#. Статические поля и методы

- Назначение статических полей класса и статических методов одинаковое и в С++ и в Java и в С#.
- Статические поля класса - это поля, значения которых для каждого экземпляра класса (объекта) одинаковы (создаются статические поля отдельно от создания объекта), а статические методы предназначены для работы со статическими полями. Статический метод ссылке this не получает и поэтому обращаться к нестатическим членам класса не может. Для доступа к обычному члену класса необходимо создать объект класса, а для доступа к статическому элементу можно воспользоваться именем класса.
- Так как в С# и в Java отсутствуют глобальные переменные и константы, то все объявления должны находиться внутри классов. В результате часто образуются классы, состоящие исключительно из статических членов. Необходимость в создании экземпляров у таких классов полностью отсутствует, так как статические поля и методы можно вызвать через имя класса. Чтобы запретить создание экземпляров данного класса создают закрытый конструктор (см. пример на слайде).

C#. Передача параметров в методы

- **Описание метода:**

Модификатор доступа Тип_возврата имяМетода (Формальные параметры)

```
{ /* тело метода */ }
```

- **Вызов метода:**

имяОбъекта . имяМетода (Фактические параметры)

- **Способы передачи параметров:**

- по значению;
- по ссылке (по адресу).

- **Разновидности формальных и фактических параметров:**

- входные параметры (параметры – значения);
- выходные параметры (объявляются с ключевым словом **out**);
- ссылочные параметры (объявляются с ключевым словом **ref**).

C#. Передача параметров в методы

- В C# существует два способа передачи параметров: по значению и по ссылке.
- При передаче параметров по значению - значения фактических параметров копируются в соответствующие формальные параметры. Изменение формального параметра не влияет на значение фактического параметра. При передаче ссылок на объекты копируется ссылка (адрес объекта). После такого копирования, и фактический параметр, и формальный параметр, ссылаются на один и тот же объект.
- При передаче параметров по ссылке - адреса фактических параметров копируются в соответствующие формальные параметры и по этим адресам есть доступ к значениям фактических параметров и их можно изменить.
- В C# передача параметров по ссылке реализуется с помощью ссылочных параметров (ref – параметров) и выходных параметров (out – параметров).
- Разница между ref и out в том, что переменную, переданную через out вы обязаны проинициализировать в методе, куда она передается.

C#. Передача параметров в методы

● Пример 2. Параметры-значения

● Вариант 1

```
using System;
namespace Primer2
{
    class Test
    {
        static void swap(int i1, int i2)
        {
            int c = i1; i1 = i2; i2 = c;
        }
        public static void Main(String[] args)
        {
            int a = 10; int b = 20;
            swap(a, b);
            Console.WriteLine("a = {0} b = {1}", a, b);
            Console.ReadKey();
        }
    }
}
```

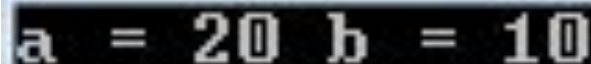
```
a = 10 b = 20
```

C#. Передача параметров в методы

● Пример 2. Параметры - ссылки

● Вариант 2

```
using System;
namespace Primer2
{
class Test
{
public static void swap(ref int i1, ref int i2)
{ int c = i1; i1 = i2; i2 = c; }
public static void Main()
{ int a = 10; int b = 20;
  swap(ref a, ref b);
  Console.WriteLine("a = {0} b = {1}", a, b);
  Console.ReadKey();
}
}
}
```



a = 20 b = 10

C#. Передача параметров в методы

● Пример 3. Массив в качестве параметра и возвращаемого значения

```
using System;
namespace Primer2
{ class Test
  { public static int[] input()
    { int n; //количество спортсменов
      string st;
      Console.Write("Укажите количество спортсменов: ");
      st = Console.ReadLine(); n = Convert.ToInt32(st);
      int[] a = new int[n];
      Console.WriteLine("Введите время заплыва каждого из {0} спортсменов: ",n);
      for (int i = 0; i < n; i++)
        a[i] = Convert.ToInt32(Console.ReadLine());
      return a;
    }
  }
```

C#. Передача параметров в методы

● Пример 3. Массив в качестве параметра и возвращаемого значения

```
public static void minim(int[] a)
{
    int min = a[0];
    for (int i = 0; i < a.Length; i++)
        if (a[i] < min)
            min = a[i];
    Console.WriteLine("Лучшее время заплыва: {0}", min);
}

public static void Main()
{
    int[] a = input();
    minim(a);
    Console.ReadKey();
}
}
```

```
Укажите количество спортсменов: 5
Введите время заплыва каждого из 5 спортсменов:
7
4
5
2
9
Лучшее время заплыва: 2
```

C#. Передача параметров в методы

● Пример 4. Массив в качестве параметра

```
using System;
namespace Primer2
{
    class Test
    {
        public static void input(int[] a)
        {
            Console.WriteLine("Введите время заплыва каждого из {0} спортсменов: ", a.Length);
            for (int i = 0; i < a.Length; i++)
                a[i] = Convert.ToInt32(Console.ReadLine());
        }

        public static void minim(int[] a)
        {
            int min = a[0];
            for (int i = 0; i < a.Length; i++)
                if (a[i] < min)
                    min = a[i];
            Console.WriteLine("Лучшее время заплыва: {0}", min);
        }
    }
}
```


C#. Передача параметров в методы

● Пример 4. Массив в качестве параметра

```
public static void Main()
{
    int n; //количество спортсменов
    string st;
    Console.WriteLine("Укажите количество спортсменов: ");
    st = Console.ReadLine(); n = Convert.ToInt32(st);
    int[] a = new int[n];
    input(a);
    minim(a);
    Console.ReadKey();
}
}
```

```
Укажите количество спортсменов: 5
Введите время заплыва каждого из 5 спортсменов:
7
4
5
2
9
Лучшее время заплыва: 2
```

C#. Передача параметров в методы

- **Пример 5.** Возврат двух значений из метода. Выходные параметры

```
using System;
namespace Primer2
{
    class Test
    {
        public static int maxValue(int[] m, out int numValues)
        {
            int result = m[0]; numValues = 0;
            for (int i = 0; i < m.Length; i++)
            {
                if (m[i] == result)
                    numValues++;
                if (m[i] > result)
                {
                    result = m[i];
                    numValues = 1;
                }
            }
            return result;
        }
    }
}
```

C#. Передача параметров в методы

- **Пример 5.** Возврат двух значений из метода. Выходные параметры

```
public static void Main()
{
    int[] m = new int[] {1, 5, 3, 5, 2};
    int numValues;
    int result = maxValue(m, out numValues);
    Console.WriteLine("Maximum value {0}. Found {1} times", result, numValues);
    Console.ReadKey();
}
}
```

```
Maximum value 5. Found 2 times_
```

C#. Массивы объектов. Интерфейс IComparable

● Пример 6. Сортировка массива объектов

```
using System;
namespace Primer1
{
    class Dog : IComparable
    {
        private string name; //кличка
        private int age; //возраст
        public string getName(){return name;}
        public int getAge() { return age; }
        public void setName(string newName) { name = newName; }
        public void setAge(int newAge) { age = newAge; }
        public Dog() { name = "NoName"; age = 1; }
        public Dog(string name, int age)
        {
            this.name = name; this.age = age; }
        public void voice()
        {
            // Console.Write("{0}: ", name);
            for (int i = 1; i <= age; i++)
                Console.Write("рав ");
            Console.WriteLine();
        }
        public int CompareTo(object obj)
        {
            Dog tmp = obj as Dog;
            // return this.name.CompareTo(tmp.name);
            if (tmp.getAge() < age) return 1;
            else if (tmp.getAge() > age) return -1;
            else return 0;
        }
    }
}
```

C#. Массивы объектов. Интерфейс IComparable

● Пример 6. Сортировка массива объектов

```
class Program
{
    static void Main(string[] args)
    {
        Dog[] dogs = {
            new Dog("Шарик", 3),
            new Dog("Тузик", 2),
            new Dog("Пушок", 5)};

        Array.Sort(dogs);
        for(int i = 0; i < dogs.Length; i++)
        {
            Console.WriteLine("{0} - возраст: ", dogs[i].getName());
            dogs[i].voice();
        }
        Console.ReadKey();
    }
}
```

```
Тузик - возраст: гав гав
Шарик - возраст: гав гав гав
Пушок - возраст: гав гав гав гав гав
```

```
Пушок - возраст: гав гав гав гав гав
Тузик - возраст: гав гав
Шарик - возраст: гав гав гав
```

● Пример 7. Сортировка массива объектов

```
using System;
using System.Collections.Generic;
namespace Primer1
{
    class Dog : IComparable <Dog>
    {
        private string name; //личка
        private int age; //возраст
        public string getName(){return name;}
        public int getAge() { return age; }
        public void setName(string newName) { name = newName; }
        public void setAge(int newAge) { age = newAge; }
        public Dog() { name = "NoName"; age = 1; }
        public Dog(string name, int age)
        {
            this.name = name; this.age = age; }
        public void voice()
        {
            // Console.WriteLine("{0}: ", name);
            for (int i = 1; i <= age; i++)
                Console.WriteLine("рав ");
            Console.WriteLine();
        }
        public int CompareTo(Dog tmp)
        {
            if (tmp.getAge() < age) return 1;
            else if (tmp.getAge() > age) return -1;
            else return 0;
        }
    }
}
```

● Пример 7. Сортировка массива объектов

```
class Program
{
    static void Main(string[] args)
    {
        Dog[] dogs = new Dog[3];
        dogs[0] = new Dog("Шарик", 3);
        dogs[1] = new Dog("Тузик", 2);
        dogs[2] = new Dog("Пушок", 5);
        Array.Sort(dogs);
        for(int i = 0; i < dogs.Length; i++)
        {
            Console.WriteLine("{0} - возраст: ", dogs[i].getName());
            dogs[i].voice();
        }
        Console.WriteLine();
        Array.Sort(dogs, new SortByName());
        for (int i = 0; i < dogs.Length; i++)
        {
            Console.WriteLine("{0} - возраст: ", dogs[i].getName());
            dogs[i].voice();
        }
        Console.ReadKey();
    }
}
```

C#. Массивы объектов. Интерфейс IComparer

● Пример 7. Сортировка массива объектов

```
1 class SortByName : IComparer<Dog>
  {
1   public int Compare(Dog obj1, Dog obj2)
   { return obj1.GetName().CompareTo(obj2.GetName()); }
   }
}
```

```
Тузик - возраст: гав гав
Шарик - возраст: гав гав гав
Пушок - возраст: гав гав гав гав гав

Пушок - возраст: гав гав гав гав гав
Тузик - возраст: гав гав
Шарик - возраст: гав гав гав
```


С#. Структуры

- Описание структуры:

```
[ Модификатор доступа ] struct ИмяСтруктуры [ : интерфейсы ]  
  
{ // члены структуры – поля и методы  
    Модификатор доступа Тип имяПоля ;  
    Модификатор доступа Тип имяМетода (Параметры) {  
        // тело метода    }  
}
```

- Модификаторы доступа для членов структуры:

- public , internal, private ;

- Доступом по умолчанию для членов структуры является private

- Создание экземпляра структуры:

ИмяСтруктуры имяОбъекта;

ИмяСтруктуры имяОбъекта = new ИмяСтруктуры([параметры конструктора]);

- Доступ к полям и методам объекта:

имяОбъекта.имяПоля = значение;

ИмяОбъекта.имяМетода(параметры);

С#. Структуры

- Структура – это тип данных, аналогичный классу, но имеющий ряд важных отличий от него:
 - Структура является значимым, а не ссылочным типом данных, то есть экземпляр структуры располагается в стеке, а не в куче;
 - Структура не может участвовать в иерархиях наследования, она может только реализовывать интерфейсы;
 - В структуре запрещено определять конструктор по умолчанию, он определен неявно и присваивает всем ее элементам значения по умолчанию (нули соответствующего типа);
 - При описании структуры нельзя задавать значения полей по умолчанию (кроме статических полей);
 - Переопределяться (override) могут только методы, унаследованные от базового класса object;
 - В структуре запрещено определять деструкторы.
- Замечание. Строго говоря любой значимый тип С# является структурным.
- Отличия от классов обуславливают область применения структур: типы данных, имеющие небольшое количество полей, с которыми удобнее работать как со значениями, а не как со ссылками. Накладные расходы на динамическое выделение памяти для небольших объектов могут значительно снизить быстродействие программы, поэтому их эффективнее описывать как структуры, а не как классы.
- Замечание. С другой стороны, передача экземпляра структуры в метод по значению требует и дополнительного времени, и дополнительной памяти.
- При выводе экземпляра структуры на консоль выполняется упаковка, то есть неявное преобразование в ссылочный тип. Упаковка применяется и в других случаях, когда структурный тип используется там, где ожидается ссылочный. При обратном преобразовании – из ссылочного типа в структурный – выполняется распаковка.
- При присваивании экземпляров структур создается копия значений полей. То же самое происходит и при передаче структур в качестве параметра по значению. Для экономии ресурсов ничто не мешает передавать структуры в методы по ссылке с помощью ключевых слов ref или out.
- Особенно значительный выигрыш в эффективности можно получить, используя массивы объектов структур вместо массивов объектов классов. Например, для массива из 100 объектов класса создается 101 объект, а для массива объектов структур – один объект.

C#. Структуры

● Пример 8.

```
using System;
namespace Primer1
{
    struct Dog : IComparable <Dog>
    {
        private string name; //кликча
        private int age; //возраст
        public string getName() { return name; }
        public int getAge() { return age; }
        public void setName(string newName) { name = newName; }
        public void setAge(int newAge) { age = newAge; }
    }
    public Dog(string name, int age)
        { this.name = name; this.age = age; }
    public void voice()
    {for (int i = 1; i <= age; i++)
        Console.Write("рав ");
        Console.WriteLine();
    }
    public int CompareTo(Dog tmp)
    { return this.name.CompareTo(tmp.name);
    }
    public override string ToString()
    { return name + " " + age;
    }
}
```

С#. Структуры

● Пример 8.

```
class Program
{
    static void Main(string[] args)
    {
        Dog[] dogs = {
            new Dog("Шарик", 3),
            new Dog("Тузик", 2),
            new Dog("Пушок", 5)};

        Array.Sort(dogs);
        for (int i = 0; i < dogs.Length; i++)
        {
            Console.WriteLine("{0} - возраст: ", dogs[i].getName());
            dogs[i].voice();
        }

        Dog dog1 = new Dog("Каштанка", 6);
        Dog dog2 = dog1;
        dog2.setName("Жучка");
        Console.WriteLine("dog1: " + dog1);
        Console.WriteLine("dog2: " + dog2);
        Console.ReadKey();
    }
}
```

```
Пушок - возраст: гав гав гав гав гав
Тузик - возраст: гав гав
Шарик - возраст: гав гав гав
dog1: Каштанка 6
dog2: Жучка 6
```

Контрольные вопросы

1. Понятие класса. Синтаксис описания класса в С#. Модификаторы доступа: характеристика и примеры использования.
2. Понятие объекта. Создание и уничтожение объектов в С#. Доступ к полям и методам объекта в С#. Конструкторы: назначение и типы. Примеры.
3. С#. Передача параметров простых и ссылочных типов в методы. Примеры.
4. С#. Статические поля и методы: назначение и примеры использования.
5. Массивы объектов в С#. Сортировка массива объектов. Примеры.
6. С#. Структуры: назначение, отличие от классов, примеры использования.