Основы C#

Бестужев Никита Евгеньевич
Преподаватель

Содержание лекции

1. Первое консольное приложение на C#
2. Объявление и инициализация переменных
3. Внутренние типы данных и операция «new»
4. Иерархия классов типов данных
5. Итерационные конструкции в C#
6. Конструкции принятия решений
7. Методы и модификаторы параметров
8. Массивы в C#
9. Типы структур
10. Типы значения и ссылочные типы

Первое консольное приложение на C#

Пример «Hello World»:

```
namespace SimpleCSharpApp
{
    class Program
    {
     static void Main(string [] args)
     {
            // Вывод простого сообщения пользователю.
            Console.WriteLine("Hello World!");
            // Ожидание нажатия клавиши <Enter>
            Console.ReadLine();
     }
    }
}
```

Первое консольное приложение на C#

Важное замечание:

C# является чувствительным к регистру языком программирования. Следовательно, Main и main или Readline и ReadLine будут представлять собой далеко не одно и то же.

Поэтому необходимо запомнить, что все ключевые слова в C# вводятся в нижнем регистре (например, public, lock, class, dynamic), а названия пространств имен, типов и членов всегда начинаются (по соглашению) с заглавной буквы, равно как и любые вложенные в них слова (как, например, Console.WriteLine, System.Windows.Forms.MessageBox и System.Data.SqlClient).

Первое консольное приложение на C#

Обработка аргументов командной строки:

```
static void Main(string[ ] args)
{
   // Обработка любых входящих аргументов.
   for(int i = 0; i < args.Length; i++)
    Console.WriteLine ("Arg: {0}", args[i]);
   Console.ReadLine();
}
```

Первое консольное приложение на C#

Класс System.Console:

Член	Описание
`Beep()`	Этот метод вынуждает консоль подавать звуковой сигнал определенной частоты и длительности
`BackgroundColor` `ForegroundColor`	Эти свойства позволяют задавать цвет изображения и фона для текущего вывода. В качестве значения им может присваиваться любой из членов перечисления `ConsoleColor`
`BufferHeight` `BufferWidth`	Эти свойства отвечают за высоту и ширину буферной области консоли
`Title`	Это свойство позволяет устанавливать заголовок для текущей консоли

`WindowHeight`
`WindowWidth`
`WindowTop`
`WindowLeft`
	Эти свойства позволяют управлять размерами консоли по отношению к установленному буферу

`Clear()`	Этот метод позволяет очищать установленный буфер и область изображения консоли

Объявление и инициализация переменных

Пример:

```
// Локальные переменные объявляются и инициализируются
// следующим образом:
// типДанных имяПеременной = начальноеЗначение;
int myInt = 0;
// Объявлять локальные переменные и присваивать им
// начальные значения можно также в двух отдельных
    строках.
string myString;
myString = "This is my character data";
Console.WriteLine("Your data: {0}, {1}", myInt, myString);
// Объявление трех переменных типа bool в одной строке.
bool b1 = true, b2 = false, b3 = b1;
```

Внутренние типы данных и операция «new»

Пример:

```
// Использование ключевого слова
// new для создания переменных
bool b = new bool (); // Установка в false.
int i = new int(); // Установка в 0.
double d = new double() ; // Установка в 0.
DateTime dt = new DateTime(); // Установка в
    1/1/0001 12:00:00 AM
Console.WriteLine ("{0}, {1}, {2}, {3}", b, i, d, dt);
```

Иерархия классов типов данных

Важное замечание:

Даже элементарные типы данных в .NET имеют вид иерархии классов. Каждый из них наследуется от класса System.Object (в котором содержится набор методов, таких как ToString(), Equals())

Иерархия классов типов данных

Числовые типы данных:

```
// Минимальное значение типа int
Console.WriteLine ("Min of int: {0}", int.MinValue);
// Максимальное значение типа double
Console.WriteLine ("Max of double: {0}",
   double.MaxValue);
```

Иерархия классов типов данных

Тип System.Boolean:

- Единственными значениями, которые могут присваиваться типу Boolean (bool) в C#, являются true и false

Иерархия классов типов данных

Тип System.Char:

- string позволяет представлять непрерывный набор символов («Hello»), а char – только конкретный символ в типе string (‘H’)
- Пример: *char myChar = ‘A’;*

Иерархия классов типов данных

Тип System.DateTime:

```
// Этот конструктор принимает в качестве
// аргументов сведения о годе, месяце и дне.
DateTime dt = new DateTime (2013, 10, 17);
// Какой это день месяца?
Console.WriteLine("The day of {0} is {1}", dt.Date,
   dt.DayOfWeek);
```

Иерархия классов типов данных

Работа со строковыми данными:

Член	Описание
`Length`	Свойство, которое возвращает длину текущей строки
`Compare()`	Статический метод, который позволяет сравнить две строки
`Contains()`	Метод, который позволяет определить, содержится ли в строке определенная подстрока
`Equals()`	Метод, который позволяет проверить, содержатся ли в двух строковых объектах идентичные символьные данные
`Format()`	Статический метод, позволяющий сформатировать строку с использованием других элементарных типов данных (например, числовых данных или других строк) и обозначений типа `{0}`, о которых рассказывалось ранее в этой главе

Итерационные конструкции в C#

Конструкции для выполнения итераций:

- Цикл for
- Цикл foreach/in
- Цикл while
- Цикл do/while

Итерационные конструкции в C#

Цикл for:

```
for(int i = 0; i < 4; i + + )
{
   Console.WriteLine("Number is: {0} " , i) ;
}
```

Итерационные конструкции в C#

Цикл foreach:

```
string[ ] carTypes = {"Ford", "BMW", "Yugo",
   "Honda" };
foreach (string c in carTypes)
   Console.WriteLine(c);
```

Итерационные конструкции в C#

Цикл while и do/while:

```
string userIsDone = "";
while(userIsDone.ToLower () != "yes")
{
   Console.Write("Are you done? [yes] [no]: ") ;
   // запрос окончания
   userIsDone = Console.ReadLine ();
   Console.WriteLine ("In while loop");
}
```

Конструкции принятия решений

Конструкции:

- Оператор if/else
- Оператор switch

Конструкции принятия решений

Оператор if/else:

В отличие от языков C и C++, в C# этот оператор может работать только с булевскими выражениями

Операция сравнения	Пример использования	Описание
==	`if(age == 30)`	Возвращает `true`, только если выражения одинаковы
!=	`if("Foo" != myStr)`	Возвращает `true`, только если выражения разные

<
>
<=
>=
	`if(bonus < 2000)`
	`if(bonus > 2000)`
	`if(bonus <= 2000)`
	`if(bonus >= 2000)`
		Возвращает `true`, только если выражение слева (`bonus`) меньше, больше, меньше или равно либо больше или равно выражению справа (`2000`)

Конструкции принятия решений

Оператор switch:

```
Console.WriteLine ("1 [C#], 2 [VB]");
Console.Write("Please pick your language preference: ");
string n= Console.ReadLine ();
switch (n)
{
    case "1": Console.WriteLine("Good choice, C# is a fine language.");
    break;
    case "2": Console.WriteLine("VB: OOP, multithreading, and more!");
    break;
    default: Console.WriteLine("Well…good luck with that!");
    break;
}
```

Методы и модификаторы параметров

Формат метода:

```
class Program
{
   //Статические методы могут вызываться
   //напрямую без создания экземпляра класса.
   static int Add(int x, int y)
   {
    return x + y;
   }
}
```

Методы и модификаторы параметров

Модификатор параметра	Описание
(отсутствует)	Если параметр не сопровождается модификатором, предполагается, что он должен передаваться по значению, т.е. вызываемый метод должен получать копию исходных данных
out	Выходные параметры должны присваиваться вызываемым методом (и, следовательно, передаваться по ссылке). Если параметрам out в вызываемом методе значения не присвоены, компилятор сообщит об ошибке
ref	Это значение первоначально присваивается вызывающим кодом и при желании может повторно присваиваться в вызываемом методе (поскольку данные также передаются по ссылке). Если параметрам ref в вызываемом методе значения не присвоены, компилятор никакой ошибки генерировать не будет
params	Этот модификатор позволяет передавать в виде одного логического параметра переменное количество аргументов. В каждом методе может присутствовать только один модификатор params и он должен обязательно указываться последним в списке параметров. В реальности необходимость в использовании модификатора params возникает не особо часто, однако он применяется во многих методах внутри библиотек базовых классов

Методы и модификаторы параметров

```
//По умолчанию аргументы передаются по значению.
public static int Add(int x, int y) {
    int ans = x + y;
    x = 10000; y = 88888;
    return ans; }
// Передача двух переменных по значению.
static void Main(string[] args) {
    int x = 9, y = 10;
    Console.WriteLine("Before call: X: {0}, Y: {1}", x, y) ; // до вызова
    Console.WriteLine("Answer is: {0}", Add(x, y) ) ; // ответ
    Console.WriteLine("After call: X: {0}, Y: {1}", x, y) ; // после вызова
    Console.ReadLine(); }
```

РЕЗУЛЬТАТ:

Before call: X: 9, Y: 10

Answer is: 19

After call: X: 9, Y: 10

Методы и модификаторы параметров

Модификатор out:

```
// Выходные параметры должны предоставляться вызываемым методом.
public static void Add (int x, int y, out int ans)
    {  ans = x + y;  }
// Присваивать первоначальное значение локальным
// переменным, используемым в качестве выходных
// параметров, не требуется, при условии, что в первый раз
// они используются в качестве выходных аргументов.
static void Main(string[] args)
{
    int ans;
    Add(90, 90, out ans);
    Console.WriteLine("90 + 90 = {0}", ans);
    Console.ReadLine();
}
```

Методы и модификаторы параметров

Модификатор ref:

```
// Ссылочные параметры.
public static void SwapStrings(ref string s1, ref string s2) {
    string tempStr = s1;
    s1 = s2;
    s2 = tempStr; }
static void Main(string[ ] args) {
    string s1 = "Flip";  string s2 = "Flop";
    Console.WriteLine("Before: {0}, {1} ", s1, s2) ; //до
    SwapStrings (ref s1, ref s2);
    Console.WriteLine("After: {0}, [1} ", s1, s2); // после
    Console.ReadLine (); }
```

РЕЗУЛЬТАТ:

Before: Flip, Flop

After: Flop, Flip

Методы и модификаторы параметров

Модификатор params:

```
// Возвращение среднего из некоторого количества значений double.
static double CalculateAverage(params double[] values)
{
    // Вывод количества значений
    Console.WriteLine ("You sent me {o} doubles.", values.Length);
    double sum = o;
    if(values.Length == o)
      return sum;
    for (int i = o; l < values.Length; i++)
      sum += values [i];
    return (sum/values.Length);
}
```

На заметку!

Во избежание какой бы то ни было неоднозначности, в C# требуется, чтобы в любом методе поддерживался только один аргумент params, который должен быть последним в списке параметров.

Методы и модификаторы параметров

Перегрузка методов:

```
class Program
{
    static void Main(string[] args) { }
    // Перегруженный метод Add() .
    static int Add(int x, int y)
    { return x + y; }
    static double Add(double x, double y)
    { return x + y; }
    static long Add(long x, long y)
    { return x + y; }
}
```

Массивы в C#

```
static void SimpleArrays ()
{
    // Создание и заполнение массива тремя
    // целочисленными значениями.
    int[] myInts = new int[3];
    myInts[0] = 100;
    myInts[1] = 200;
    myInts[2] = 300;
    // Отображение значений.
    foreach (int i in myInts)
      Console.WriteLine(i);
    Console.WriteLine();
}
```

Массивы в C#

```
static void ArrayInitialization ()
{
    // с помощью  ключевого слова new.
    string[] stringArray = new string[] { "one", "two", "three" };
    Console.WriteLine("stringArray has {0} elements",
    stringArray.Length);
    // без применения  ключевого слова new.
    bool [] boolArray = { false, false, true };
    Console.WriteLine("boolArray has {0} elements", boolArray.Length);
    // с указанием ключевого  слова new и желаемого размера.
    int[] intArray = new int[4] { 20, 22, 23, 0 };
    Console.WriteLine("intArray has {0} elements", intArray.Length);
    Console.WriteLine();
}
```

Массивы в C#

Передача массива в качестве аргумента:

```
static void PrintArray (int [] myInts)
{
for(int i = 0; i < myInts.Length; i++)
    Console.WriteLine("Item {0} is {1} ", i, myInts[i]);
}
static string[] GetStringArray()
{
    string[] theStrings = {"Hello", "from",
    "GetStringArray"};
    return theStrings;
}
```

Массивы в C#

Член класса System.Array	Описание
Clear()	Статический метод, который позволяет устанавливать для всего ряда элементов в массиве пустые значения 0 – для чисел, null – для объектных ссылок и false – для булевских выражений)
CopyTo()	Метод, который позволяет копировать элементы из исходного массива в целевой
Length	Свойство, которое возвращает информацию о количестве элементов в массиве
Rank	Свойство, которое возвращает информацию о количестве измерений в массиве
Reverse()	Статическое свойство, которое представляет содержимое одномерного массива в обратном порядке
Sort()	Статический метод, который позволяет сортировать одномерный массив внутренних типов.

Типы структур

На заметку!

Если вы ранее занимались объектно-ориентированным программированием, можете считать структуры "облегченными классами", поскольку они тоже предоставляют возможность определять тип, поддерживающий инкапсуляцию, но не могут применяться для построения семейства взаимосвязанных типов. Когда есть потребность в создании семейства взаимосвязанных типов через наследование, нужно применять типы классов.

В C# структуры создаются с помощью ключевого слова struct.

Типы структур

```
struct Point  {
    // Поля структуры.
    public int X;
    public int Y;
    // Добавление 1 к позиции (X, Y) .
    public void Increment ()
    { X++; Y++;  }
    // Вычитание 1 из позиции (X, Y) .
    public void Decrement ()
    { X--; Y—;  }
    // Отображение текущей позиции.
    public void Display()
    { Console.WriteLine(X = {0}, Y= {1}", X, Y);  }
}
```

Типы значения и ссылочные типы

```
// Локальные структуры извлекаются из стека
// после завершения метода.
static void LocalValueTypes ()
{
   //В действительности int представляет
   // собой структуру System.Int32.
   int i = 0;
   //В действительности Point представляет
   // собой тип структуры.
   Point p = new Point();
} // Здесь i и p изымаются из стека.
```

Типы значения и ссылочные типы

```
static void ValueTypeAssignment() {
    Point p1 = new Point (10, 10); Point p2 = p1;
    // Вывод обеих переменных Point,
    p1.Display();  p2.Display();
    // Изменение значение p1.X и повторный вывод.
    // Значение p2.X не изменяется.
    p1.X = 100;
    Console.WriteLine("\n=> Changed p1.X\n");
    p1.Display(); p2.Display(); }
```

РЕЗУЛЬТАТ:

```
X = 10, Y = 10
X = 10, Y = 10
=> Changed p1.X
X = 100, Y = 10
X = 10, Y = 10
```

Типы значения и ссылочные типы

```
class PointRef  {
    public PointRef(int XPos, int YPos)
    { X = XPos;  Y = YPos;  }
}
static void ReferenceTypeAssignment ()  {
    PointRef p1 = new PointRef (10, 10);   PointRef p2 = p1;
    p1.Display();  p2.Display();
    p1.X = 100;
    Console.WriteLine("\n=> Changed p1.X\n");
    p1.Display();  p2.Display();  }
```

РЕЗУЛЬТАТ:

```
X = 10, Y = 10
X = 10, Y = 10
=> Changed p1.X
X = 100, Y = 10
X = 100, Y = 10
```

Передача ссылочных типов по значению и по ссылке

Важные отличия:

- В случае передачи ссылочного типа по ссылке вызывающий код может изменять значения данных состояния объекта, а также сам объект, на который указывает входная ссылка.
- В случае передачи ссылочного типа по значению вызывающий код может изменять только значения данных состояния объекта, но не сам объект, на который указывает входная ссылка.

Спасибо за внимание!