Modern Programming Technologies

Lecture 2. C# language

Victor Babkov
PhD, Applied Mathematics & Informatics dept.
Victor.Babkov@gmail.com

Первая программа

Задача программы вывести на экран строку

«Hello, World »

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

namespace ConsoleTest
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(“Hello, World");
        }
    }
}
```

Пространства имен

```
namespace ConsoleTest
{
  …
}
```

- Упрощают программирование и структурируют код

Директива using

- Размещается в начале файла
- Подставляет префиксы пространств имен

Пример

```
using System;
Console.WriteLine(“Hello, World”);
```

Или

```
System.Console.WriteLine(“Hello, World”);
```

Элементы программирования

- Комментарии
- Типы данных
- Декларация переменных
- Операторы
- Циклы
- Классы

Базовые операции

Групповой оператор

```
{
    …
}
```

Пустой оператор

```
;
```

Комментарии

```
//  комментарий

/*
   комментарий
*/

/// <summary>
///
/// </summary>
```

Базовые типы C#

- Целые знаковые sbyte, short, int, long (1,2,4,8)
- Целые беззнаковые byte, unsigned short (ushort), unsigned int (uint), unsigned long (1,2,4,8)
- С плавающей точкой double, float
- С фиксированной точкой decimal
- Символьный char
- Строковый string
- Логический boolean

Диапазоны

sbyte	-127	128
byte	0	255
short	-32768	32767
ushort	0	65535
int	-2147483648	2147483647
uint	0	4294967295
long	-9223372036854775808	9,223,372,036,854,775,807
ulong	0	18446744073709551615
char	0	65535
boolean	false	true

Базовые типы - классы

- long -> System.Int64
- short -> System.Int16
- int -> System.Int32
- string -> System.String
- char -> System.Char
- ushort -> System.UInt16

Операции

- математические
- логические
- битовые

Математические

+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Остаток от деления

Класс Math

Abs	Модуль	PI	Число Пи
Acos	Арккосинус	Pow	Возведение в степень
Asin	Арксинус	Sign	Знак числа
Atan	Арктангенс	Sin	Синус
Cos	Косинус	Sinh	Гиперболический синус
Cosh	Гиперболический синус	Sqrt	Корень
E	Основание натурального логарифма	Tan	Тангенс
Exp	Експонента	Tanh	Гиперболический тангенс
Log	Логарифм с произв. основанием	**Truncate**	Отбрасываение дробной части
Log10	Десятичный логарифм	**Round**	Округление с указанной точностью
Max	Максимум из двух чисел	**Ceiling**	Ближайшее большее целое
Min	Минимум из двух чисел	**Floor**	Ближайшее меньшее целое

Логические операторы

&&	«AND»
||	«OR»
==, <=, >=,<,>	Сравнение
!	Инверсия

Битовые операторы

&	«AND»
|	«OR»
^	«XOR»
~	Инверсия
>>, <<	Сдвиг

5&4=4

5|6=7

5^6=3

Переменные

```
int I;

I = 20;

int  m =20;

const int a = 100;
```

Помощь компилятора

```
int i;
int n;
n = i*i; Ошибка!
```

```
class Program
{
    static void Main(string[] args)
    {
        int i, j;
        i = 20; j = i; j = 10;
        Console.WriteLine(i);
        Console.WriteLine(j);
    }
}
```

Область видимости

- Поле, член класса существует пока существует объект
- Локальная переменная существует до тех пор, пока не появится скобка }, которая закрывает блок, в котором определена переменная

Работа с консолью

- вывод на консоль

```
Console.WriteLine("test");
```

- Управление параметрами окна (цвет, размер, заголовок);
- Управление курсором;
- Форматированный вывод.

Форматированный вывод

```
Console.WriteLine("{0},{1},{2},{0}",
  10, 20, 30)
```

На экране:

10,20,30,10

После : можна указать формат:

{0:формат} C, E, F, D, X, ... после формата могут быть параметры

После запятой можно указать ширину поля:

{0, 4} – выравнивание вправо

{0, -10} – выравнивание влево

Ввод из консоли

```
string s = Console.ReadLine();
```

Есть методы для:

- ввода кода нажатой клавиши Readkey
- Для определения нажатия CapsLock, NumLock

Преобразование в базовые типы

```
string s = Console.ReadLine();
double d = Double.Parse(s);
```

Управляющие структуры

- Операторы ветвления if / else
- Операторы выбора switch { case … }
- оператор цикла for (;;) {}
- оператор цикла while () {}
- оператор цикла do { } while ()

Операторы ветвления if / else

```
if (i>0)
{
 Console.WriteLine(“positive”);
}
else if (i<0)
 Console.WriteLine(“negative”);
else
 Console.WriteLine(“zero”);
```

Операторы выбора

```
switch ( <выражение> )
{
    case <значение 1>:
      …
        break;
    case <значение 2>:
      …
        break;
    default:
      …
}
```

Циклы

В C# есть 3 типа циклов:

- `for ( ; ; ) { }`
- `while () { … }`
- `do { … } while ();`

Цикл for(;;)

```
for ( <Начальное д-вие>
;
 <условие окончания>
;
<д-вие на каждом шаге>
)
{
}
```

Классический цикл for

```
for (int i=0; i<20; i++)
{
    Console.WriteLine(i);
}

for (short i=0; i<20; i--)
{
    Console.WriteLine(i);
}
```

Разные примеры цикла

- Бесконечный цикл

```
for ( ; ; );
```

```
for ( ;true; )
{
}
```

Выход из цикла for

```
string s = Console.ReadLine();
int i = Int32.Parse(s);
for ( ;; i++)
{
    if ( i % 127 == 0)
      break;
    Console.WriteLine(i);
}
```

Переход к след. итерации

```
string s = Console.ReadLine();
int i = Int32.Parse(s);
for ( ;(i%127 != 0);i++ )
{
    if ( i % 2 == 0)
      continue;
    Console.WriteLine(i);
}
```

Цикл while {...}

```
int i = 20;
while (i<50)
{
    Console.WriteLine(i);
    i++;
}
```

Цикл do {…} while;

```
do
{
    Console.WriteLine(i);
    i++;
}
while (i<50);
```

Массивы

- Определение массива
- Инициализация массива
- Работа с массивами
- Многомерные массивы

Декларация и инициализация

```
int[] myarr;

myarr = new int[5];
```

Доступ к элементам

```
int[] myarr = new int[7];
for (int i=0; i< myarr.Length;
  i++)
{
   Console.WriteLine(myarr[i]);
}
```

Заполнение массива

```
int[] myarr = new int[7];
for (int i=0; i< myarr.Length;
  i++)
{
   myarr[i] = i*i;
}
```

Копирование массива

```
int[] myarr = new int[6];
int[] myarrcopy = new int[7];
for (int i=0; i< myarr.Length; i++)
{
  myarrcopy[i] = myarr[i];
}
myarrcopy[6] = new int();
myarr = myarrcopy;
```

Декларация и инициализация

```
int[] arr = {1,2,3,4,5};
```

или

```
int[] arr = new int[5];
arr[0] = 1;
```

…

```
arr[4] = 5;
```

Многомерные массивы

```
int[,] arr ;
arr = new int[3,2];
arr[1,1] =5;

int[,,] bb ;
bb = new int[3,2,10];
bb[1,1,1] =5;
```

Декларация и инициализация

```
int[,] a =
  {
      { 1,2,3,4},
      { 1,4,5,8},
      { 1,2,3,6},
      { 1,7,8,9}
  };
```

Массивы – объекты со свойствами и методами

- Length – длина массива
- GetLength – длинна в указанном измерении
- Rank – кол-во измерений

Для одномерных

Max

Min

Sum

Структуры

struct <имя структуры>
{
  <тип поля> <имя поля>;
}

```
struct apple
{
    int Massa;
   int Color;
}
```

Класс

Состоит из:

- полей
- методов
- свойств
- конструкторов

Перечисления(enum)

```
enum Weekday { SUNDAY, MONDAY,
  TUESDAY};

Weekday.SUNDAY;
```

Поля в классах

```
class apple
{
    public int Massa;
    public string Sort;
}
```

Пример

```
class apple    {
      public int Massa;
      public string Sort;
}
  class Program
  {
      static void Main(string[] args)
      {
          apple a;
         a = new apple();
         a.Massa = 10;
         a.Sort = "antonovka";
         int i = a.Massa + 10;
      }
  }
```

Декларация методов

```
class <имя класса>
{
   public <тип результата> Massa (
  <тип1> <имя_параметра1>, … , <типN>
  <имя_параметраN>…)
   {
      int I;
      …
      return I;
   }
}
```

Примеры

```
class Test
{
  public void Method1()
  {
  }
  public void Method2(int i)
  {
  }
  public int Method3( long I )
  {
     int I;
     …
     return I;
  }
}
```

Параметры ref, out

```
int a;
obj.Method1(0,a);

void Method1(int i, int a)
{
  a=0;
}
```

Параметры ref, out

```
int a;    Нужно int a =0;
obj.Method1(0,ref a);

void Method1(int i, ref int a)
{
  a=0;
}
```

Параметры ref, out

```
int a;
obj.Method1(0,out a);

void Method1(int i, out int a)
{
  a=0;
}
```

Перегрузка методов (операторов)

```
class Test
{
  public void MakeSth()
  {
  }
  public void MakeSth(int i)
  {
  }
}
```

Ключевое слово this

```
public Apple
{
    int Massa;
    public void MakeHalf(int Massa)
    {
        this.Massa = Massa;
    }
}
```

Конструктор

- Метод без типа возврата и без return
- Его имя совпадает с именем класса
- Может быть несколько конструкторов
- Если не указывать, то у каждого класса будет конструктор по умолчанию

```
class Apple
   {
      public int Massa;
      public Apple()
      {
         Console.WriteLine("Ura!");
      }
   }

   class Program
   {
      static void Main(string[] args)
      {
         Apple a;
         a = new Apple();
      }
   }
```

Ошибка, нет конструктора по умолчанию

```
class Apple
{
        public int Massa;
        public Apple (int I ) { this.Massa = I;
  };
}
class Program
{
      static void Main(string[] args)
       {
            Apple a;
            a = new Apple();
        }
}
```

Свойства

```
class Apple {
   private int massa;
   public int Massa    {
      get {
          return massa;
      }
      set  {
         if (value > 0 )
          this.massa = value;
      }
   }
}
```

Модификаторы доступа

Основые:

- public (доступно всем пользователям класса)
- protected (доступно классу и его потомкам)
- private (доступно только самому классу)

Дополнительно:

- sealed
- virtual, override
- partial

Наследование

```
public class ChildObject :
  ParentObject
```

При этом у нового объекта автоматически появляются public і protected поля, методы и свойства унаследованного класса

Множественного наследования нет!!!

Переопределение методов

```
class ParentObject {
virtual void ParentMethod(
  <параметры>)
..
}
class ChildObject {
override void ParentMethod(
  <параметры>)
..
}
```

Вызов метода базового класса

```
Class C1
{public virtual void method1()
 {}
}
Class C2 : C1
{public override void method1()
{base.method1();}
}
```

Поля readonly

Если поле должно быть неизменным, но его значение определяется на этапе запуска программы, то поле можно определить как readonly

readonly int MyConst;

Присвоить значение можно только в конструкторе

Статические поля и методы

```
Class C1
{static public int a;
  public int b;
}
C1 Ob1 = new C1();
C1 Ob2 = new C1();
Ob1.b  та Ob2.b – это разные поля
Ob1.a та Ob2.a – это одно и то же поле
Можно даже  C1.a.
```

Интерфейси

Интерфейс показывает, что должен делать класс, но не определяет как.

```
public interface IBankAccount
{
void PayIn(decimal amount);
bool withdraw(decimal amount);
decimal Balance
  {get ;}
}
```

```
public class SaverAccount: IBankAccount
{
 private decimal balance;

 public void PayIn(decimal amount)
    {balance+=amount;}

 public bool Withdraw(decimal amount)
   {if(balance>=amount)
      {balance-=amount;
      return true;
      }
   else
      return false;
   }

  public decimal Balance
   {get{return balance;}}
}
```

Аналогично можно создать другой класс с таким же интерфейсом, но другой реализацией :

```
public class GoldAccount: IBankAccount
{
 private decimal balance;

 public void PayIn(decimal amount)
    {другая реализация}

 public bool Withdraw(decimal amount)
   {другая реализация}

  public decimal Balance
   {get{другая реализация}}
}
```

Как использовать интерфейсы

```
IBankAccount account1 = new SaverAcount();
IBankAccount account2 = new GoldAccount();


IBankAccount[] accounts= new IBankAccount[2];
accounts[0] = new SaverAccount();
accounts[1] = new GoldAccount();
```

Множественное наследование через интерфейсы

```
class MyClass: I1, I2, I3 , ….
{}

class MyClass: C1, I1, I2 , ….
{}

Class MyClass C1,C2,… - нельзя
{}
```

Сравнение ссылочных типов

Метод ReferenceEquals(p1,p2) – статический метод, который есть всегда и который возвращает true, если ссылки указывают на один и тот же адрес

```
MyClass x,y;
x = new MyClass();
y = new MyClass();
bool B1 = ReferenceEquals(null,null); //true
bool B2 = ReferenceEquals(null, x); // false
bool B3 = RefrenceEquals(x,y); //false
bool B4 = ReferenceEquals(x,x); //true
```

Сравнение ссылочных типов

Equals(p1,p2) – виртуальный метод,который можно переопределить по потребностям.

Сравнение ссылочных типов

== для ссылочных типов сравнивает ссылки,
но оператор можно перегрузитьь.

Структурированные хранилища информации

- Массивы
- Коллекции
- Обобщения

Коллекции

- Обычные коллекции содержат тип object
- Обобщения могу хранить конкретный тип данных

Для использования нобходимо подключить System.Collections;

ArrayList

Динамический массив

```
Пример создания
//по умолчанию размер - 16
ArrayList ar = new ArrayList();

//конкретный размер - 10
ArrayList ar = new ArrayList(10);
```

ArrayList - методы

Add
AddRange
BinarySearch
Clear
Contains
GetRange
IndexOf
Insert
InsertRange
Remove
RemoveAt
RemoveRange
Reverse
Sort
TrimToSize

ArrayList

Свойства:

Capacity – емкость

Count – реальное кол-во элементов

ArrayList

Если реальный размер превышает емкость, то емкость удваивается

Для прохода по коллекциям можно использовать два вида циклов

```
ArrayList Ar = new ArrayList();
…
for(int I =0;i<Ar.Count;i++)
    Console.WriteLine(Ar[i]);

foreach (object ob in Ar)
    Console.WriteLine(ob);
```

Пример работы с ArrayList

```
string s="fsf sf sfs0 fsf";
ArrayList ar = new ArrayList();
string[] words = s.Split(' ');

for(int i =0;i<words.Length;i++)
  if(!ar.Contains(words[i]))
      ar.Add(words[i]);
int maxlen=0;
int imax=-1;

for(int i=0;i<ar.Count;i++)
    if (((string)ar[i]).Length > maxlen)
    {   maxlen = ((string)ar[i]).Length;
        imax = i;
    }
ar.RemoveAt(imax);
Console.WriteLine(imax);
int sum=0;

foreach(object ob in ar)
  {sum+=((string)ob).Length;
  }
Console.WriteLine(sum);
```

SortedList

Динамический массив, в котором каждый элемент – это пара (ключ - значение)

```
SortedList sl = new SortedList();

sl.Add(“Victor”, 30);// “victor” – ключ, 30 - значение
sl.Add(“Jane”, 25);
sl.Add(“John”,45);

Console.WriteLine(sl[“Victor”]); // 30
```

SortedList - методы

Add
Clear
ContainsKey
ContainsValue
GetByIndex
GetKey
GetKeyList
GetValueList
IndexOfKey
IndexOfValue
Remove
RemoveAt

SortedList

Проход по элементам SortedList

```
SortedList sl = new SortedList();
…
foreach(object ob in sl.GetKeyList())
{
    Console.WriteLine(sl[ob]);
}
```

Stack, Queue

Для стека:

Clear
Contains
Push
Pop
Peak
Count

Для очереди:

Clear
Contains
Dequeue
Enqueue
Peek
Count

Hashtable, Dictionary

Изучить самостоятельно

Обобщения

Позволяют явно указать тип, т.е. работают безопаснее и быстрее простых коллекций

List<тип> l = new List<тип>();

Обобщения

List<>

SortedList<>

LinkedList<>

Queue<>

Stack<>

Делегаты

Делегат – указатель на функцию (то, что дает имя сигнатуре)

```
delegate void MyDelegate(int x);
```

MyDelegate – это указатель на любой метод, который не возвращает результат и имеет один параметр типа int

Пример использования делегатов

Например, мы реализуем метод для сортировки пузырьком

```
class BubbleSorter
        {
            static public void Sort(int[] SortArray)
            {
                for(int i =0;i<SortArray.Length;i++)
                    for (int j = i + 1; j<SortArray.Length; j++)
                    {
                        if (SortArray[j] < SortArray[i])
                        {
                            int temp = SortArray[i];
                            SortArray[i] = SortArray[j];
                            SortArray[j] = temp;
                        }
                    }
            }
        }
```

Метод не позволит сравнивать произвольные типы

Пример использования делегатов

```
class BubbleSorter
    {
       public delegate bool CompareOp(object ob1, object ob2);

        static public void Sort(object[] SortArray,CompareOp method)
       {
          for(int i =0;i<SortArray.Length;i++)
             for (int j = i + 1; j<SortArray.Length; j++)
             {
                if (method(SortArray[j],SortArray[i]))
                {
                   object temp = SortArray[i];
                   SortArray[i] = SortArray[j];
                   SortArray[j] = temp;
                }
             }
       }
    }
```

Пример использования делегатов

```
public delegate bool CompareOp(object ob1, object ob2);
```

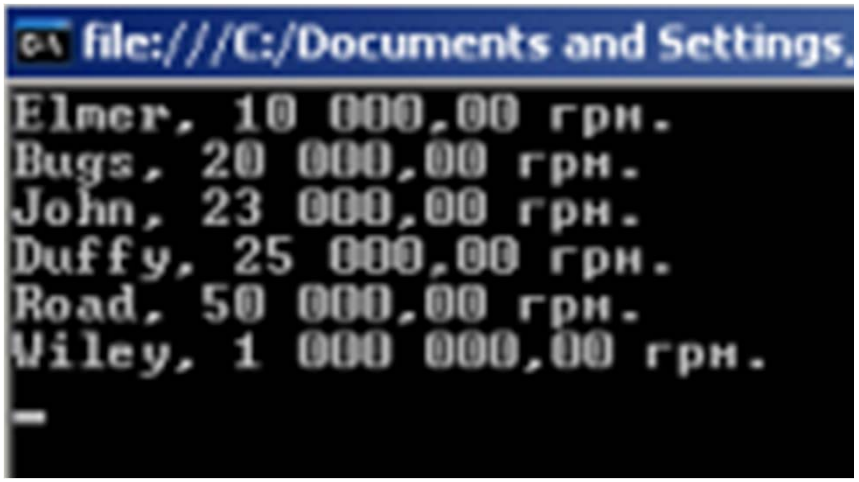
Эта строка показывает, что вместо CompareOp можно использовть любой другой метод, который возвращает true, если первый параметр “меньше”второго.

Пример использования делегатов

```
class Employee
        {
            private string name;
            private decimal salary;

            public Employee(string name, decimal salary)
            {
                this.name = name;
                this.salary = salary;
            }

            public override string ToString()
            {
                return string.Format(name + ", {0:C}", salary);
            }

            public static bool Greater(object ob1, object ob2)
            {
                Employee e1 = (Employee)ob1;
                Employee e2 = (Employee)ob2;
                return (e2.salary > e1.salary) ? true : false;
            }
        }
```

Пример использования делегатов

```
static void Main(string[] args)
      {   Employee[] employees =
         {  new Employee("Bugs",20000),
            new Employee("Elmer",10000),
            new Employee("Duffy",25000),
            new Employee("Wiley",1000000),
            new Employee("John",23000),
            new Employee("Road",50000)
         };

         BubbleSorter.CompareOp employeeCompareOp = new
                                 BubbleSorter.CompareOp(Employee.Greater);

         BubbleSorter.Sort(employees, employeeCompareOp);
         for (int i = 0; i < employees.Length; i++)
            Console.WriteLine(employees[i].ToString());
      }
```



Групповые делегаты

```
delegate void MyDelegate(int x);
MyDelegate md = new MyDelegate(method1);
md+=new MyDelegate(method2);
md+=new MyDelegate(method3);
```

Вызов md(x) вызовет последовательно все три метода

Можно также:

```
md-=MyDelegate(method3);
```

Неявные типы (только локальные переменные)

```
// i is compiled as an int
var i = 5;

// s is compiled as a string
var s = "Hello";

// a is compiled as int[]
var a = new[] { 0, 1, 2 };

// expr is compiled as IEnumerable<Customer>
// or perhaps IQueryable<Customer>
var expr =
    from c in customers
    where c.City == "London"
    select c;

// anon is compiled as an anonymous type
var anon = new { Name = "Terry", Age = 34 };

// list is compiled as List<int>
var list = new List<int>();
```

Конструкции выборки

```
// Example #1: var is optional because
// the select clause specifies a string
string[] words = { "apple", "strawberry", "grape", "peach", "banana" };
var wordQuery = from word in words
                where word[0] == 'g'
                select word;

// Because each element in the sequence is a string,
// not an anonymous type, var is optional here also.
foreach (string s in wordQuery)
{
    Console.WriteLine(s);
}

// Example #2: var is required because
// the select clause specifies an anonymous type
var custQuery = from cust in customers
                where cust.City == "Phoenix"
                select new { cust.Name, cust.Phone };

// var must be used because each item
// in the sequence is an anonymous type
foreach (var item in custQuery)
{
    Console.WriteLine("Name={0}, Phone={1}", item.Name, item.Phone);
}
```

Анонимные методы

C#

```
// Create a handler for a click event
button1.Click += delegate(System.Object o, System.EventArgs e)
                    { System.Windows.Forms.MessageBox.Show("Click!"); };
```

or

C#

```
// Create a delegate instance
delegate void Del(int x);

// Instantiate the delegate using an anonymous method
Del d = delegate(int k) { /* ... */ };
```

Вопросы?