

# Введение в C#

# План занятия

- Отличие C# от C++
- Типы данных
- Базовые выражения и операторы
- Программа Hello, C# world
  - Консольное приложение
  - Windows приложение
- Исключения

# НОВЫЙ ЯЗЫК ОТ Microsoft, Intel, HP

- Язык на котором написана платформа .NET Framework
- Результат эволюции языков программирования
- C# - “на пол тона выше” C++

А впрочем, можете программировать на Java 😊

# Некоторые отличия C# от C++

- В C# отсутствует множественное наследование (компенсируется мн. насл. интерфейсов)
- Переменные любого типа (даже bool и int) являются объектами
- Для освобождения памяти используется система сборки мусора среды выполнения
- Невозможно определить глобальные переменные или методы, их определения должны находиться внутри классов

# Некоторые отличия C# от C++

- Все данные делятся на ссылочные и размерные
- Ссылочные хранятся в общем пуле памяти, а размерные могут храниться в стеке метода
- Данные типа `bool` могут принимать только два значения `true` и `false`, при этом не допускается преобразование этого типа в другие
- Разрядность каждого типа данных не зависит от ОС и процессора

# Основные отличия C# от C++

- Использование указателей ограничено областями *небезопасного кода*. Вместо указателей на функции используется механизм специальных методов – *делегатов*
- В C# структуры являются размерными типами данных, а классы – ссылочными
- Добавлены новые операторы и ключевые слова *is*, *as*, *ref*, *out*, *foreach*. В операторе *try-catch* добавлен блок *finally*

# Типы данных в C#

# Целые типы

Название типа	Диапазон значений	Размер
<b>sbyte</b>	<b>-128..127</b>	<b>Знаковое 8-битное целое</b>
<b>byte</b>	<b>0..255</b>	<b>Беззнаковое 8-битное целое</b>
<b>char</b>	<b>U+0000..U+FFFF</b>	<b>16-битный Unicode-символ</b>
<b>short</b>	<b>-32.768..32.767</b>	<b>Знаковое 16-битное целое</b>
<b>ushort</b>	<b>0..65535</b>	<b>Беззнаковое 16-битное целое</b>
<b>int</b>	<b>-2.147.483.648..2.147.483.647</b>	<b>Знаковое 32-битное целое</b>
<b>uint</b>	<b>0..4.294.967.295</b>	<b>Беззнаковое 32-битное целое</b>
<b>long</b>	<b>-9.223.372.036.854.775.808.. 9.223.372.036.854.775.807</b>	<b>Знаковое 64-битное целое</b>
<b>ulong</b>	<b>0..18.446.744.073.709.551.615</b>	<b>Беззнаковое 64-битное целое</b>



- Типы с плавающей точкой

<b>Название типа</b>	<b>Диапазон значений</b>	<b>Точность</b>
<b>Float</b>	<b>+1.5E-45..+3.4E38</b>	<b>7 знаков</b>
<b>Double</b>	<b>+5.0E-324..+1.7E308</b>	<b>15-16 знаков</b>

- Тип decimal

<b>Название типа</b>	<b>Диапазон значений</b>	<b>Точность</b>
<b>Decimal</b>	<b>1.0E-28 to 7.9E28</b>	<b>28-39 знаков</b>

# Структуры

- Сходны с классами. Отличие: являются типом значения
  - Структура передаются по значению, а не по ссылке
- Объявление структуры подобно объявлению класса:

```
public struct Point
{
    public int x, y;
    public Point(int p1, int p2)
    { x = p1; y = p2; }
}
```

# Одномерные массивы

- Объявление:

```
int[ ] arraysize=10;
```

```
int[ ] array0 = new int[arraysize];
```

```
int[ ] array1 = {1, 2, 3, 4, 5};
```

- Доступ к элементам:

```
int element = array1[0];
```

- Нумерация индексов от 0 до N - 1, N - размер массива

# Многомерные массивы

- Элементы многомерных массивов идентифицируются набором индексов - "координат" в многомерном пространстве
- Объявление:  

```
int[,] array = new int[10, 20, 30];  
int[,] array = {{1, 2}, {3, 4}};
```
- Доступ к элементам:  

```
int element = array[0, 1, 2];
```

# Неровные (jagged) массивы

Неровные массивы – это массивы массивов (ссылок на массивы)

Декларация:

```
int array[ ] [ ] = new int[2];  
array[0] = new int[4];  
array[1] = new int[20];
```

Доступ:

```
int element = array[0][1];
```

- ! Размеры неровных массивов могут различаться даже в одном измерении. В приведенном примере существует элемент `array[1][15]`, но не существует элемента `array[0][15]`

# Все значения являются объектами

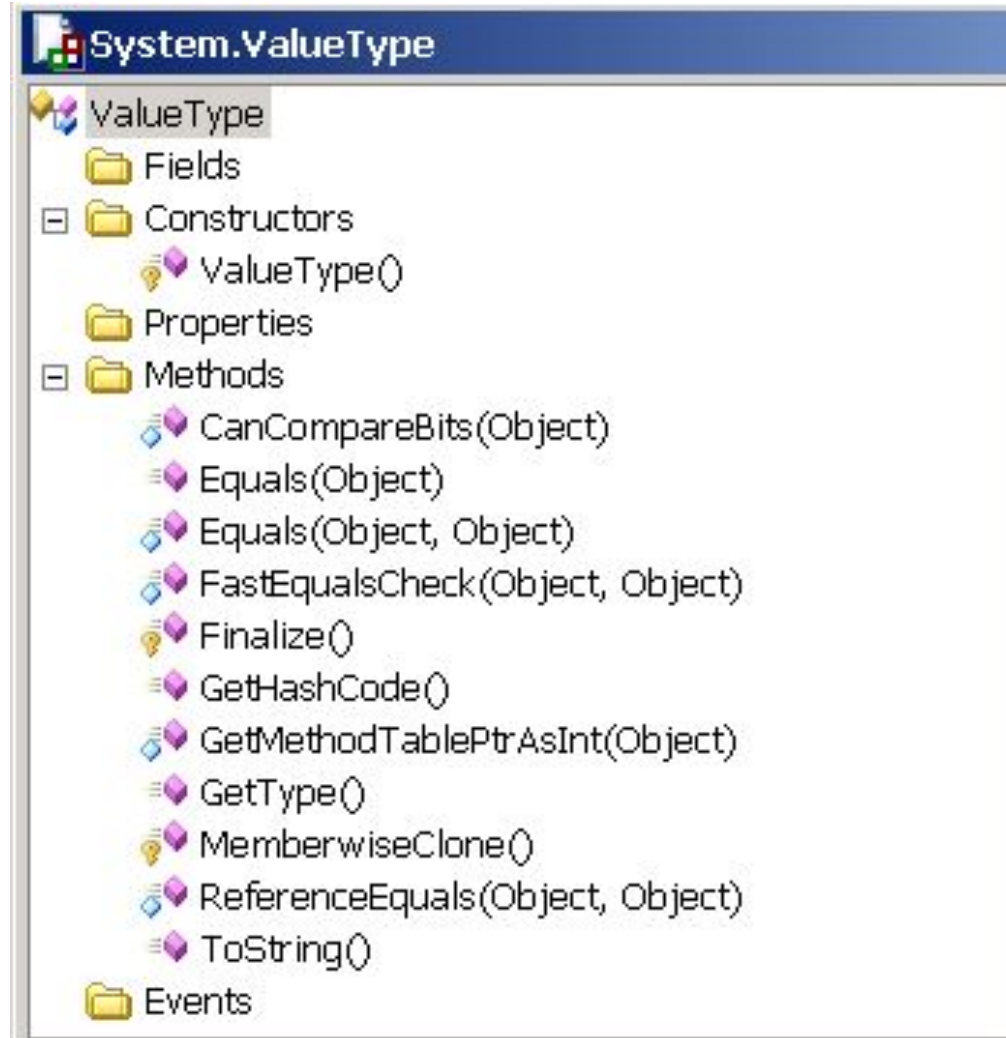
- Все типы значений представлены соответствующими типами .NET Framework из пространства имен System. Они наследуются от класса ValueType
- Для каждого значения поддерживается соответствующий "упакованный" (boxed) тип, который является классом, реализующим то же поведение и содержащим те же данные
- Если требуется передать значение по ссылке, оно автоматически упаковывается (box) в соответствующий упакованный тип, а по прибытии при необходимости распаковывается (unbox). Находясь в упакованном виде, тип может использовать все методы класса System.ValueType

Например, допустима следующая конструкция:

```
string s = 2004.ToString();
```

!

# Окно Matrix ClassBrowser для класса ValueType



# Классы наследники от ValueType

System.Object

System.ValueType

- System.Byte
- System.Char
- System.Decimal
- System.Boolean
- System.Double
- System.Int16
- System.Single
- System.DateTime
- System.Enum
- System.Void



# Базовые выражения и операторы C#

# Операторы

Арифметические	+ - * / %
Логические (булевые, побитовые)	&   ^ ! ~ &&    true false
Катенация строк	+
Инкремент, декремент	++ --
Сдвиг	<< >>
Отношение	== != < > <= >=

# Операторы (продолжение)

Присваивание	= += -= *= /= %= &=  = ^= <<= >>=
Доступ к члену	.
Индексация	[]
Условие	?:
Катенация и добавление делегата	+ -

# Операторы (продолжение)

Создание объекта	new
Информация о типе	as is sizeof typeof
Контроль переполнения	checked unchecked
Адресация	* -> [] &

# Математические операторы

<b>Символ</b>	<b>Оператор</b>
<b>+</b>	<b>Сложение</b>
<b>-</b>	<b>Вычитание</b>
<b>*</b>	<b>Умножение</b>
<b>/</b>	<b>Деление</b>
<b>%</b>	<b>Остаток от целоч. деления</b>

# Унарные операторы

<b>Символ</b>	<b>Оператор</b>
<b>+</b>	<b>Унарный плюс</b>
<b>-</b>	<b>Унарный минус</b>
<b>++</b>	<b>Инкремент</b>
<b>--</b>	<b>Декремент</b>
<b>!</b>	<b>Унарное логическое отрицание</b>
<b>~</b>	<b>Унарная поразрядная операция дополнения</b>
<b>(...)</b>	<b>Преобразование типа выражения</b>

# Составные операторы

<b>Символ</b>	<b>Оператор</b>
<b>+=</b>	<b>Сложение и присваивание</b>
<b>-=</b>	<b>Вычитание и присваивание</b>
<b>*=</b>	<b>Умножение и присваивание</b>
<b>/=</b>	<b>Деление и присваивание</b>
<b>%=</b>	<b>Вычисление остатка от деления и присваивание</b>
<b>&amp;=</b>	<b>Логическое И и присваивание</b>
<b> =</b>	<b>Логическое ИЛИ и присваивание</b>
<b>^=</b>	<b>Логическое ИСКЛЮЧАЮЩЕЕ ИЛИ и присваивание</b>
<b>&lt;&lt;=</b>	<b>Сдвиг влево и присваивание</b>
<b>&gt;&gt;=</b>	<b>Сдвиг вправо и присваивание</b>

# Поразрядные операторы

<b>Символ</b>	<b>Оператор</b>
<b>&amp;</b>	<b>Логическое либо побитовое И</b>
<b> </b>	<b>Логическое либо побитовое ИЛИ</b>
<b>^</b>	<b>Логическое либо побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ</b>
<b>~</b>	<b>Унарная поразрядная операция дополнения</b>
<b>&lt;&lt;</b>	<b>Сдвиг влево</b>
<b>&gt;&gt;</b>	<b>Сдвиг вправо</b>



# Логические операторы

<b>Символ</b>	<b>Оператор</b>
<b>&amp;&amp;</b>	<b>Логический оператор И</b>
<b>   </b>	<b>Логический оператор ИЛИ</b>
<b>!</b>	<b>Унарное логическое отрицание</b>

# Операторы отношения

<b>Символ</b>	<b>Оператор</b>
<b>==</b>	<b>Равно</b>
<b>!=</b>	<b>Не равно</b>
<b>&lt;</b>	<b>Меньше</b>
<b>&lt;=</b>	<b>Меньше или равно</b>
<b>&gt;</b>	<b>Больше</b>
<b>&gt;=</b>	<b>Больше или равно</b>

# Управляющие операторы

Оператор	Назначение	Пример
<b>if ...else</b>	Оператор условного перехода	<b>if (a &gt; b) Console.WriteLine ("a &gt; b"); else Console.WriteLine("a &lt;= b");</b>
<b>switch</b>	Оператор ветвления	<b>switch (a) {case 1:{Console.WriteLine("A= 1"); break;} case 2:{Console.WriteLine("A= 2"); break;} default: {Console.WriteLine("A &lt;&gt; 0 и A &lt;&gt; 1"); break;}}</b>
<b>goto</b>	Оператор безусловного перехода	<b>Exit: ... goto Exit;</b>
<b>for</b>	Оператор цикла	<b>for (int i = 0; i &lt; 10; i++) sum += array[i];</b>
<b>while</b>	Оператор цикла с предусловием	<b>while (x &gt; 0) {i++; x--;}</b>

# Управляющие операторы

Оператор	Назначение	Пример
<b>foreach</b>	Оператор цикла для работы с массивами и контейнерами	<pre><b>int [ ] nums = {2,4,8,16,32};</b> <b>foreach (int j in nums)</b> <b>{Console.WriteLine("j = {0}",j )};</b></pre>
<b>do</b>	Оператор цикла с постусловием	<pre>string s = "A, B, C, D"; do {s=s.Substring(s.IndexOf(",")+1); Console.WriteLine("s = {0}", s );}} while (s.Length &gt; 2);</pre>
<b>continue</b>	Выполнение цикла сначала	<pre><b>for (i=0;;i++)</b> <b>{ Console.WriteLine("{0}",i);</b> <b>if(i&lt;9) continue;</b> <b>else break; }</b></pre>
<b>break</b>	Прерывание выполнения цикла	
<b>return</b>	Возврат управления из метода	

# Операторы AS, IS, TYPEOF

as	<pre>public static void Main() {     object [ ] myObjects = new object[3];     myObjects[0] = "hello"; myObjects[1] = 123; myObjects[2] = 123.4;     for (int i=0; i&lt;myObjects.Length; ++i) {string s = myObjects[i] as     string;     Console.Write ("{0}:", i);     if (s != null) Console.WriteLine ( "" + s + "" );     else Console.WriteLine ( "not a string" );}}</pre>
is	<pre>public static void Main() {     object [ ] myObjects = new object[3];     myObjects[0] = "hello"; myObjects[1] = 123; myObjects[2] = 123.4;     for (int i=0; i&lt;myObjects.Length; ++i) { Console.Write("{0}:", i);     if (myObjects[i] is string) Console.WriteLine ( "it is a string" );}}</pre>
typeof	<pre>public static void Main() {     Type t = typeof(MyClass);}     // альтернативой будет следующая конструкция     // MyClass t1 = new MyClass();     // Type t = t1.GetType();</pre>

# Трансляция программ на C#

- Визуальная среда разработки Microsoft Visual Studio.Net
- Пакетный транслятор, входящий в Microsoft.Net Framework SDK (доступен для бесплатной загрузки)
- Среда SharpDeveloper (доступна для бесплатной загрузки)

# Программа Hello, C# world!

```
using System;
namespace Hello
{
    class HelloApp
    {
        static void Main()
        {
            Console.WriteLine("Hello, C# world");
            Console.ReadLine();
        }
    }
}
```

# Запуск программы Hello, C# world

- Набираем текст программы в любом текстовом редакторе
- Сохраняем его с именем `hello.cs` в директории `C:\Temp`
- В командной строке набираем
  - `C:\...\Framework\v1.1\csc.exe hello.cs`
- Запускаем сгенерированный `hello.exe`

Наша первая программа работает!



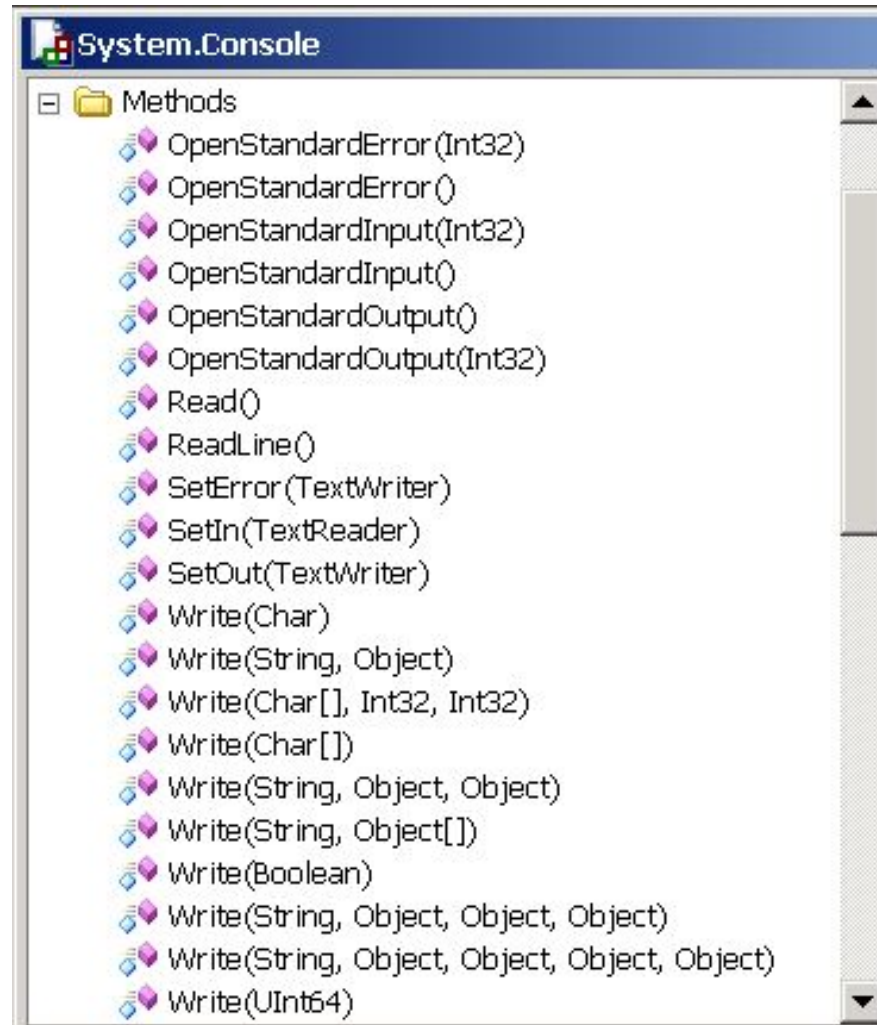
# А так наша программа выглядит на MSIL

```
.namespace 'Hello'{
  .class /*02000002*/ private auto ansi beforefieldinit 'HelloApp'
    extends ['mscorlib'/* 23000001 */]'System'.Object/* 01000001 */
  {
    .method /*06000001*/ private hidebysig static
      void 'Main()' cil managed { .entrypoint .maxstack 1
IL_0000: /* 72 | (70)000001 */ ldstr "Hello, C# world" /* 70000001 */
IL_0005: /* 28 | (0A)00000D */ call void ['mscorlib'/* 23000001 */]'System'.Console/* 0100000E
*/::'WriteLine'(string) /* 0A00000D */
IL_000a: /* 28 | (0A)00000E */ call string ['mscorlib'/* 23000001 */]'System'.Console/*
0100000E */::'ReadLine'() /* 0A00000E */
IL_000f: /* 26 | */ pop
IL_0010: /* 2A | */ ret
      } // end of method 'HelloApp'::'Main'
    .method /*06000002*/ public hidebysig specialname rtspecialname
      instance void .ctor() cil managed {
      // Method begins at RVA 0x2070
      // Code size 7 (0x7) .maxstack 1
IL_0000: /* 02 | */ ldarg.0
IL_0001: /* 28 | (0A)00000F */ call instance void ['mscorlib'/* 23000001 */]'System'.Object/*
01000001 */::'ctor'() /* 0A00000F */
IL_0006: /* 2A | */ ret
      } // end of method 'HelloApp'::'ctor'
  } // end of class 'HelloApp'
} // end of namespace 'Hello'
```

# Работа с консолью

- Для работы с консолью в .NET Framework используется класс `System.Console`
- Все его методы класса `Console` являются статическими, (не нужно создавать его экземпляр)
- Класс ответственен за ввод, вывод и вывод ошибок
- По умолчанию ввод/вывод производится на стандартную консоль, либо вообще не производится, если консоль отсутствует

# Окно Matrix ClassBrowser для класса Console



# Метод Read

- Читает символ из потока ввода
- Возвращает код прочитанного символа (значение типа `int`), либо `-1`, если ничего прочитано не было.

- Пример:

do

```
{ int i = Console.Read();  
if (i != -1) Console.WriteLine("{0} ({1})",  
(char)i, i); else break;  
} while (true);
```

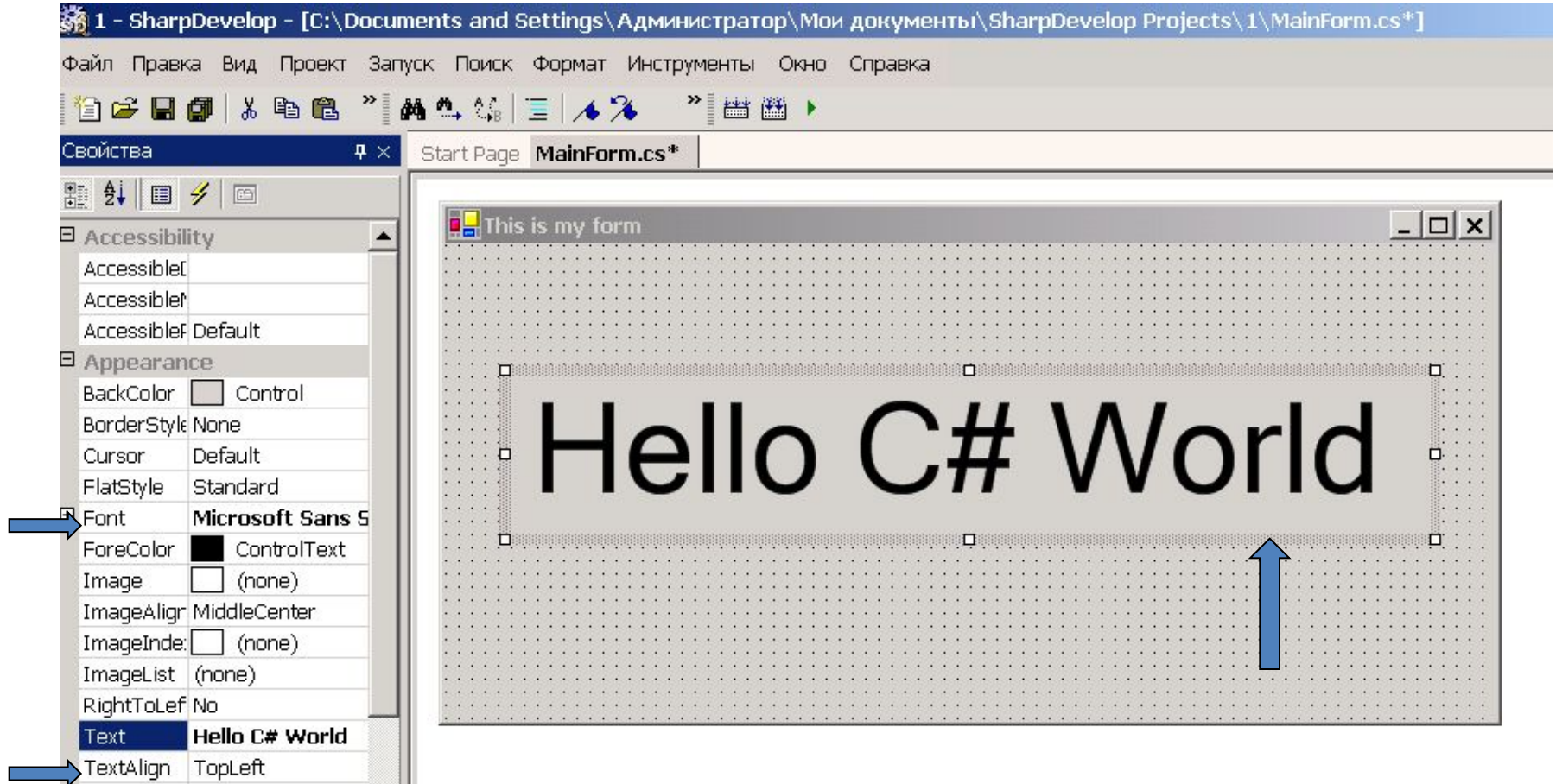
# Метод Write

- Предназначен для вывода на экран
- Определен для всех базовых типов
  - `Console.Write(1);`
  - `Console.Write(3.14159265);`
  - `Console.Write("Строка");`
- Поддерживает форматированные строки
- При форматировании может применяться ряд модификаторов, например, вместо "{n}" подставляется n-й входной параметр
  - `Console.Write("Привет, {0}", Name);`

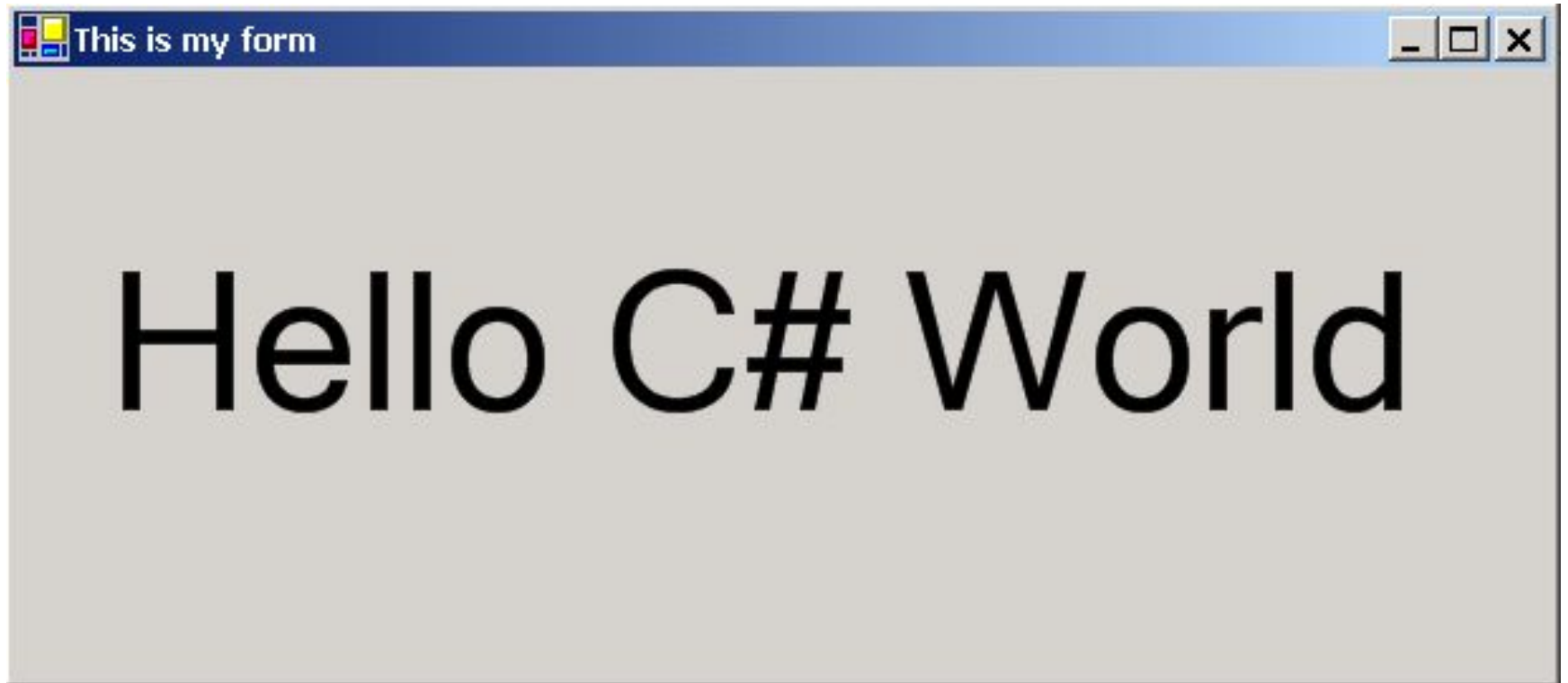
# Hello C# World с помощью SharpDevelop

- Запускаем SharpDeveloper
- Создаем новый комбинированный проект
- Выбираем тип приложения - WindowsForms
- Помещаем на поле формы Label
- Изменяем свойства Text: Hello C# World и Font:48
- Запускаем приложение

# Проект в работе



# Проект в действии





# Обработка ошибок

- Предположим, произошла ошибка и дальнейшее запланированное выполнение программы теряет смысл. Что делать?
- Можно прервать процесс и закончить работу
- Можно продолжить работу, если в коде, определена стратегия поведения в “аварийной” ситуации
- В старых языках программирования приходилось писать многоуровневые проверки с `if`, что громоздко и неудобно...

# Пример обработки ошибки деления на ноль в рамках структурного программирования

```
using System;
namespace DivideByZero {
class DivideByZero {
static void Main (string[ ] arg) {
    int i;
    for (i= -3;i<3;i++) {
        if (i==0)
            {Console.WriteLine (“Ошибка в программе – деление на
            ноль, i= {0} ”,i); i++;
            int j=1/i;
            Console.ReadLine();
            }
    }
}
```

Проверка

Реакция на ошибку

# Исключения

- При возникновении ошибки создается объект, который ее описывает и на его основе выбрасывается исключение
- При генерации исключения выполнение текущего кода прекращается. Идет возврат на более высокий уровень. На нем исключение может быть "поймано" и обработано
- Если исключение не обрабатывается, идет переход на уровень выше и т.д. пока не дойдем либо до обработчика, либо до системного вызова, в котором все исключения обрабатываются

# Синтаксис исключений

try

*{ блок команд, в котором может возникнуть ошибка }*

*[catch [(тип\_исключения имя\_исключения)]*

*{ блок обработки исключения }]*

*[finally { команды, которые выполняются в любом случае }]*

# Пример

```
using System;
namespace DivideByZeroEX {
class DivideByZeroEXApp {
static void Main (string[ ] arg){
int i=0;
try { int j=1/i;}
catch (System.Exception ex) {
Console.WriteLine (“Ошибка в программе {0} [{1}]\n\n{2}”, ex.Source,
ex.Message, ex.StackTrace); }
Console.ReadLine();
}}
```

Ссылка на объект  
класса System.Exception

Имя  
программы

Текст  
сообщения

Строка

# Еще один пример

```
class ReverseFunction {
    public static double Calculate( double d )
    { if (d == 0) throw new Exception
      ("Function is undefined in 0");
      return 1 / d; }
    public static void Main() {
    try
    {Console.WriteLine("1 / {0} = {1}", 2, Calculate(2));
      Console.WriteLine("1 / {0} = {1}", 0, Calculate(0));
      Console.WriteLine("1 / {0} = {1}", 1, Calculate(1)); }
    catch (Exception e) { Console.WriteLine("Error:{0}", e); } }
```

Создаем свой  
объект

# И еще один пример

- Для того, чтобы обработать нужное исключение ставится несколько блоков `catch`
- Вызывается первый блок, чей тип исключения соответствует типу пришедшего исключения

```
try { CalculateSpline(); }  
catch (DivisionByZeroException e) {  
    Console.WriteLine("Divizion by 0"); }  
catch (OverflowException e) { Console.WriteLine("Overflow"); }  
catch (Exception e) //Обработка остальных типов      исключений  
    { Console.WriteLine("Exception: {0}", e); }
```

# Стандартные классы исключений

- `System.DivideByZeroException`
- `System.OverflowException`
- `System.ArrayTypeMismatchException`
- `System.ArgumentOutOfRangeException`
- `System.IO.FileNotFoundException`
- `System.OutOfMemoryException`
- `System.StackOverflowException`

*А всего их - больше сотни*



# Заключение

- Существует язык программирования разработанный с учетом особенностей технологии .NET Framework
- Для разработки приложений существуют мощные визуальные средства
- Сам язык достаточно прост для изучения

# Вопросы для закрепления материала

- Назовите два отличия C# от C++
- Почему в C# отсутствуют деструкторы классов?
- Чем транслировать программы на C#?
- Зачем необходимо собственное пространство имен?
- Зачем нужна упаковка типов?
- Какие новые типы данных Вы узнали?
- Опишите назначение оператора foreach
- Что такое исключение?

# Ссылки

- Учебник по C#
  - <http://www.dotsite.spb.ru/Tutorials/CSharp/>
- Джесс Либерти “Программирование на C#” 2е изд. СПб-М., Символ, 2003
- Троелсен Э. C# и платформа .NET. Библиотека программиста. – СПб.: Питер, 2002
- Фролов А.В., Фролов Г.В. Язык C#. Самоучитель.-М.: ДИАЛОГ-МИФИ, 2003