



ОСНОВЫ ТИПОВ И БАЗОВЫЕ ПРОГРАММНЫЕ КОНСТРУКЦИИ С#

.NET & JS LAB

Анжелика КРАВЧУК

Основы типов

Основы типов

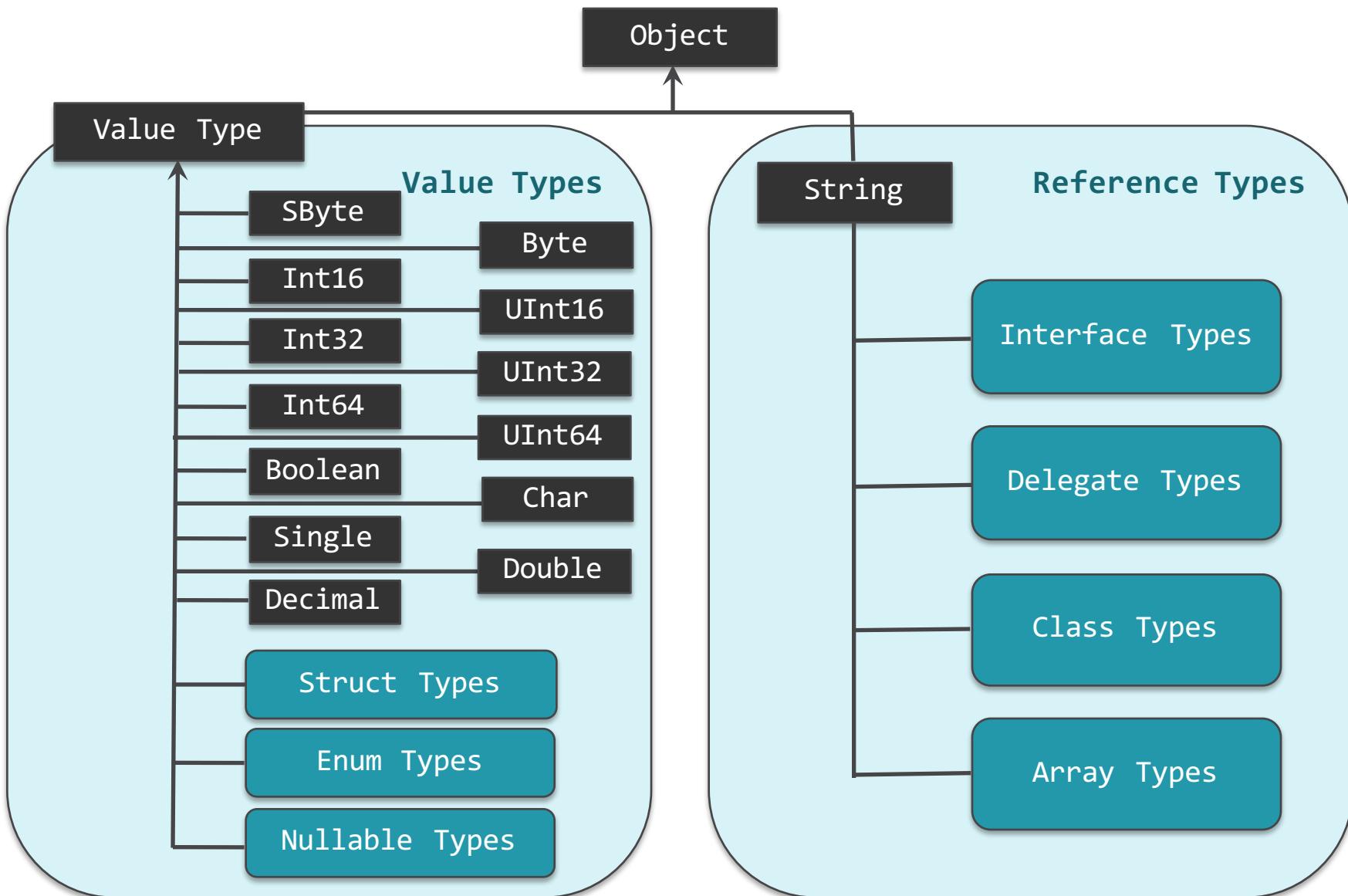
Тип – это именованная абстракция, предназначенная для повторного использования.

Языки программирования содержат синтаксические конструкции, предназначенные для создания типов (язык C# - классы, структуры, перечисления, интерфейсы, делегаты).

Каждая конструкция некоторым образом отображает принципы определения типов CLR. Тип CLR – это именованная абстракция, пригодная для повторного использования. Описание типов CLR находится в метаданных модуля CLR.

Имя типа CLR состоит из трех частей – имени сборки, необязательного префикса, обозначающего имя пространства имен и локального имени типа.

Основы типов



Определение членов типа

Определение типа CLR состоит из определения нулевого или большего количества членов типа (*members*).

От членов типа зависят способы использования и правила функционирования типа. Для каждого члена типа определен (возможно неявно) модификатор доступа, управляющий доступом к члену типа. Члены типа, доступные извне, обобщенно называются контрактом типа (*type contract*)

Кроме правил доступа к членам, можно определить, должен ли существовать экземпляр типа для доступа к его члену. Практически все члены можно разделить либо на члены экземпляра, либо на члены типа. Для обращения к члену экземпляра, должен существовать экземпляр типа

Определение членов типа

Существует три фундаментальных вида членов типа – поля, методы и вложенные типы

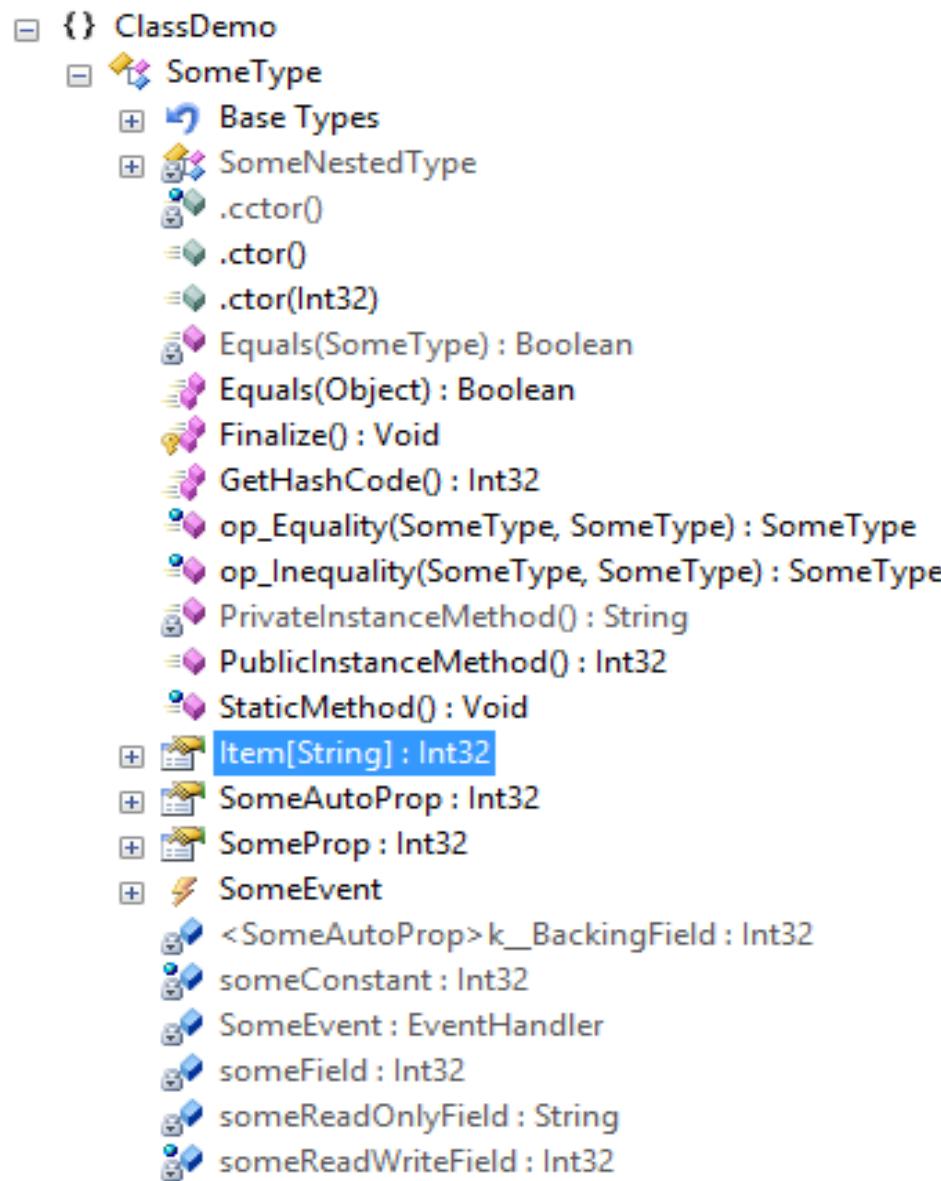
Поле – это именованная единица хранения данных, ассоциированная с типом

Метод – это именованная операция, которую можно вызвать и выполнить

Вложенный тип – это другой тип, определенный как часть реализации объявляющего типа

Другие виды членов типа – события, свойства и т.д. – это методы, расширенные с помощью метаданных!

Определение членов типа



Использование типов

Сами по себе типы используются довольно редко, полезными их делает возможность создания экземпляров. Экземпляр – это объект или значение, в зависимости от определения типа

Экземпляры значимых типов это значения, ссылочных типов – объекты

Каждый объект или каждое значение являются экземпляром только одного типа

Использование типов

Связь между типом и его экземпляром может быть явной или неявной

Объявление локальной переменной или поля типа `System.Int32` приводит к выделению в памяти блока, связанного со своим типом только посредством кода, обрабатывающего этот блок. Среда CLR (компилятор и процедура проверки CLR) обеспечивают поддержку связи этого блока памяти с типом после загрузки кода

Каждый объект связан с типом явно. Поскольку объект доступен только посредством ссылки, фактический тип ссылки объекта может не совпадать с объявленным типом ссылки. В подобных ситуациях нужен механизм, явно связывающий объект с его типом – в CLR это заголовок объекта (`object header`)

Понятие переменной

Переменная представляет именованное место в памяти для хранения порции данных

Переменная характеризуется:

Имя (Name)

Адрес (Address)

Тип данных (Data type)

Значение (Value)

Область видимости (Scope)

Время жизни (Lifetime)

Переменные ссылочного и значимого типа

```
int x1 = 42;
```

x1 : 42

System.Int32

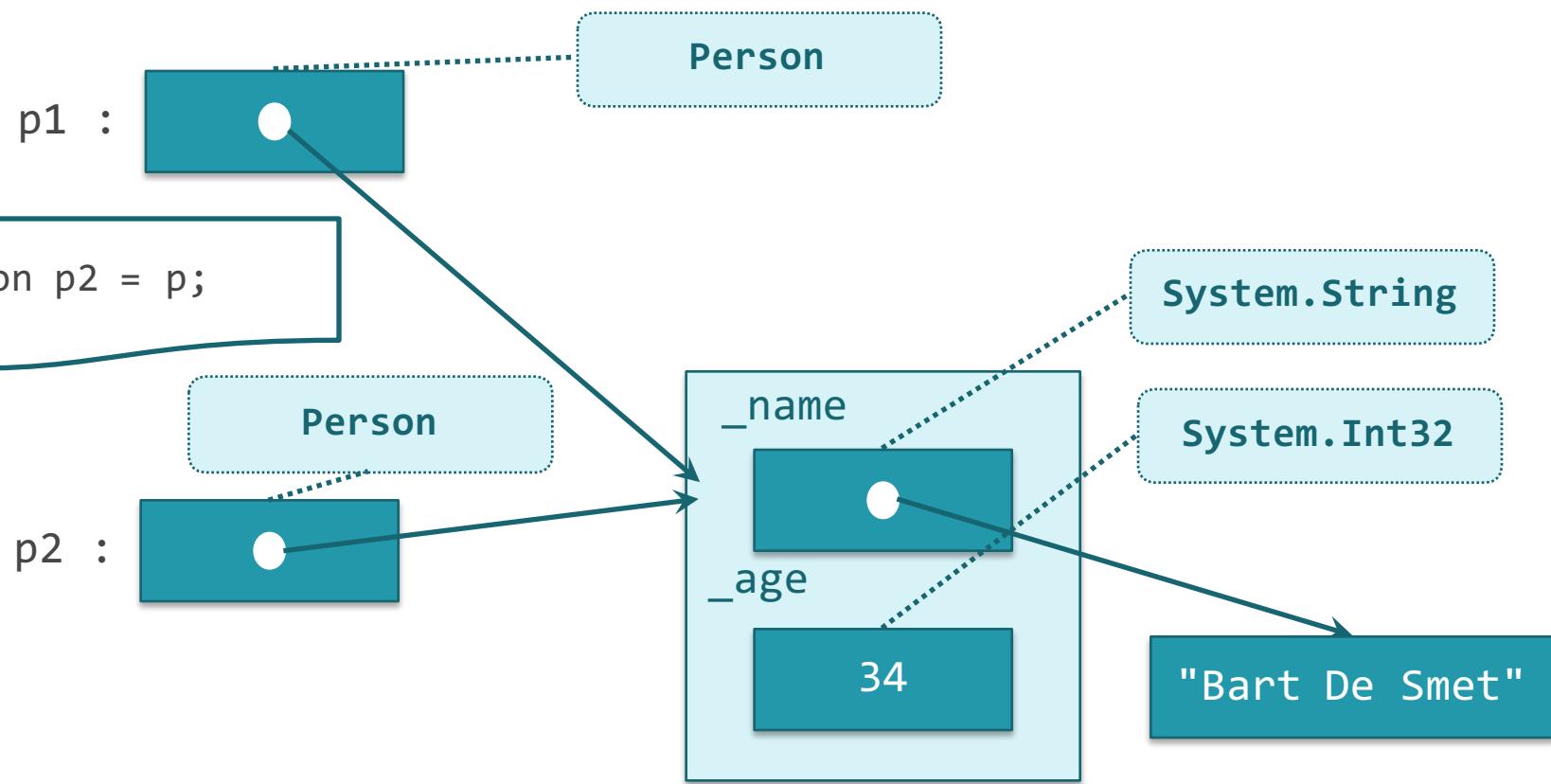
```
int x2 = x1;
```

x2 : 42

System.Int32

Переменные ссылочного и значимого типа

```
Person p1 = new Person("Bart De Smet", 34);
```



Примитивные типы C#

byte

- System.Byte
- Целое без знаковое число
- 1 байт
- 0 до 255

short

- System.Int16
- Целые числа (маленький диапазон)
- 2 байта
- -32 768 до 32 767

int

- System.Int32
- Целые числа
- 4 байта
- -2 147 483 648 до 2 147 483 647

long

- System.Int64
- Целые числа (большой диапазон)
- 8 байт
- -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807

float

- System.Single
- Числа с плавающей точкой
- 4 байта
- $+/-3,4 \times 10^38$

Примитивные типы C#

double

- System.Double
- Числа двойной точности (более точные) с плавающей точкой
- 8 байт
- $+ / -1,7 \times 10^308$

decimal

- System.Decimal
- Денежный значения
- 16 байт
- 28 значащих цифр

char

- System.Char
- Один символ
- 2 байта
- N/A

bool

- System.Bool
- Логический
- 1 байт
- true или false

Примитивные типы C#

string

- `System.String`
- Последовательность символов
- 2 байта на символ
- N/A

object

- `System.Object`
- Служит базовым классом для всех типов в мире .NET
- Позволяет сохранять любой тип в объектной переменной

Объявление и присваивание переменных

Идентификатор может содержать только буквы, цифры и символы подчеркивания

Идентификатор должен начинаться с буквы или символа подчеркивания

Идентификатор не должен быть одним из ключевых слов, которые C# резервирует для собственного использования

При объявлении переменной для ее хранения должно быть зарезервировано место в памяти, размер которого определяется типом, поэтому при объявлении переменной необходимо указать тип хранимых данных

Объявление и присваивание переменных

```
DataType variableName;  
// or  
DataType variableName1, variableName2;  
int variableName;
```

После объявления переменной можно присвоить значение для его дальнейшего использования в приложении с помощью оператора присваивания

```
variableName = value;
```

Тип выражения при присваивании должен соответствовать типу переменной

При объявлении переменной, пока ей не присвоено значение, она содержит случайное значение

```
int numberOfEmployees;  
numberOfEmployees = "Hello";
```

СТЕ

Объявление и присваивание переменных

При объявлении переменных вместо указания явного типа данных можно использовать ключевое слово var

```
var price = 20;
```

Неявную типизацию можно использовать для любых типов, включая массивы, обобщенные типы и пользовательские специальные типы

Неявная типизация применима только для локальных переменных в контексте какого-то метода или свойства

```
class ThisWillNeverCompile
{
    private var someInt = 10;
    public var MyMethod(var x, var y) { }
}
```



CTE

Объявление и присваивание переменных

Неявно типизированную локальную переменную можно возвращать вызывающему методу, при условии, что возвращаемый тип этого метода совпадает с типом, лежащим в основе определенных с помощью var данных

```
static int GetAnIntValue()
{
    var retVal = 9;
    return retVal;
}
```

Локальным переменным, объявленным с помощью ключевого слова var, не допускается присваивать в качестве начального значения null

```
var someObj = null;
```



СТЕ

Объявление и присваивание переменных

Значение неявно типизированной локальной переменной может быть присвоено другим переменным, причем как неявно, так и явно типизированным

```
var someObj = (int?)null;  
someObj = 78;
```

```
var someCar = new Car();  
someCar = null;
```

```
var someInt2 = 0;  
var anotherInt = someInt;  
string someString2 = "Wake up!";  
var someData = someString;
```

Область видимости переменной

Block scope

```
if (length > 10)
{
    int area = length * length;
```

Procedure scope

```
void ShowName()
{
    string name = "Bob";
}
```

Class scope

```
private string message;
void SetString()
{
    message = "Hello World!";
```

Namespace scope

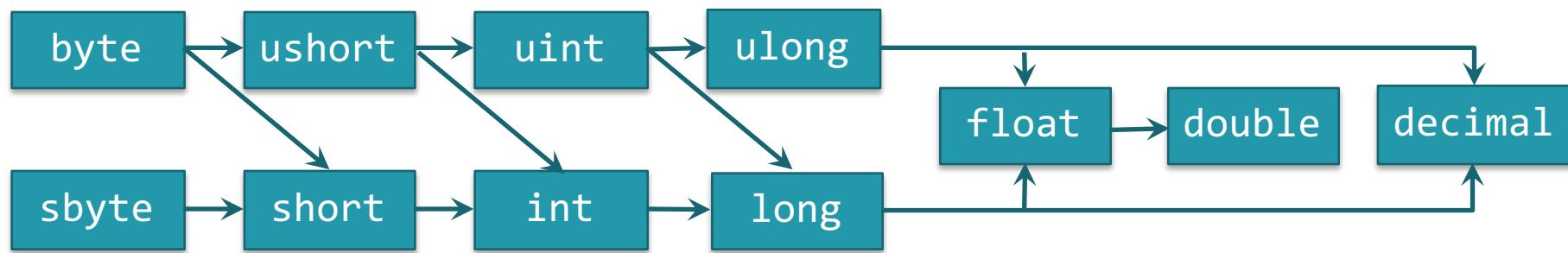
```
public class CreateMessage
{
    public string message = "Hello";
}

...
public DisplayMessage
{
    public void ShowMessage()
    {
        CreateMessage newMessage
            = new CreateMessage();
        MessageBox.Show
            (newMessage.message);
    }
}
```

Преобразование типов данных

Неявное преобразование (implicit conversion)

Не требует особых синтаксических конструкций и осуществляется компилятором



Преобразование типов данных

Явное преобразование (explicit conversion), требует операции приведение (casting)

Требует, чтобы был написан код для выполнения преобразования, которое, в противном случае, может привести к потере информации или ошибке

```
DataType variableName1 = (DataType)variableName2
. . .
string possibleInt = "1234";
int count = Convert.ToInt32(possibleInt);
. . .
int number = 1234;
string numberString = number.ToString();
. . .
int number = 0;
string numberString = "1234";
if (int.TryParse(numberString, out number))
{// Conversion succeeded, number now equals 1234}
```

Константы и переменные только для чтения

Константы

Используются только для хранения неизменяемых данных

Объявляются с помощью ключевого слова const

Значение можно инициализировать только во время разработки

```
const DataType variableName = Value;  
const double PI = 3.14159;  
int radius = 5;  
double area = PI * radius * radius;  
double circumference = 2 * PI * radius;
```

Константы и переменные только для чтения

Переменные только для чтения (read-only)

Используются только для хранения неизменяемых данных

Объявляются с помощью ключевого слова `readonly`

Значение можно инициализировать во время выполнения

```
readonly DataType variableName = Value;  
readonly string currentDateTime = DateTime.Now.ToString();
```

Выражения, операции, операторы

Использование выражений и операций в C#. Выражения

Выражения фундаментальная конструкция, используемая для вычисления и управления данными

Выражения являются комбинацией operandов и операций

a + 1

(a + b) / 2

"Answer: " + c.ToString()

b * System.Math.Tan(theta)

Использование выражений и операций в C#. Операции

Арифметические +, -, *, /, %

Инкремент, декремент ++, --

Сравнение ==, !=, <,>, <=, >=, is

Логические/битовые &, |, ^, &&, !, ||

Индексация []

Приведение (), as

Присваивание =, +=, -=, *=, =%, = . . .

Битовый сдвиг <<, >>

Информация о типе SizeOf, TypeOf

Конкатенация и удаление делегатов +, -

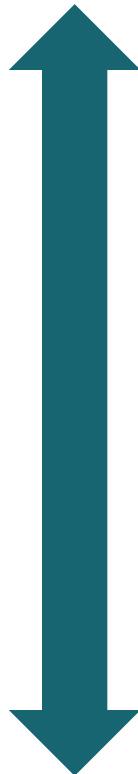
Контроль за переполнением checked, unchecked

Разыменования и получения адреса *, ->, [], &

Условная ?:

Использование выражений и операций в C#. Приоритет операций

Наивысший



Самый низкий

++, -- (префиксные), +, - (унарные), !, ~

*, /, %

+, -

<<, >>

<, >, <=, >=

==, !=

&

^

|

&&

||

операции присваивания

++, -- (постфиксные)

Оператор if. Сокращенная форма (one-way if)

Синтаксис

```
if ([condition]) [code to execute]
```

```
if ([condition])
{
    [code to execute if condition is true]
}
```

Пример

```
if (a > 50)
{
    // Add code to execute if a is greater than 50 here.
}
```

Оператор if. Сокращенная форма (one-way if)

C# предлагает две логические операции: логическая операция AND, которая представлена знаком операции «`&&`» и логическая операция OR, которая представлена знаком операции «`||`»

```
bool validPercentage;  
if (percent >= 0 && percent <= 100)  
{  
    validPercentage = true;  
}
```

```
validPercentage = percent >= 0 && percent <= 100;
```

```
bool invalidPercentage;  
if (percent < 0 || percent > 100)  
{  
    invalidPercentage = true;  
}
```

Оператор if. Полная форма (either-or if)

Синтаксис

```
if ([condition])
{
    [code to execute if condition is true]
}
else
{
    [code to execute if condition is false]
}
```

Пример

```
if (a > 50)
{
    // Add code to execute if a is greater than 50 here.
}
else
{
    // Add code to execute if a is less than or equal to 50 here.
}
```

Оператор if. Полная форма (either-or if)

Использование тернарной операции «?:» - альтернатива использованию полной формы оператора if

Синтаксис

Type result = [condition] ? [true expression] : [false expression]

Пример

```
string carColor = "green";
string response = carColor == "red" ? "You have a red car" : "You do not
have a red car";
```

Оператор if. Лесенка if else if...(multiple-outcome if)

Синтаксис

```
if ([condition])
{
    [code to execute if condition is true]
}
else if ([condition2])
{
    [code to execute if condition is false and condition2 is true]
}
else
{
    [code to execute if condition and condition2 are both false]
}
```

Оператор if. Лесенка if else if...(multiple-outcome if)

Пример

```
if (a > 50)
{
    // Add code to execute if a is greater than 50 here.
}
else if (a > 10)
{
    Add code to execute if a is greater than 10 and less than
    // or equal to 50 here.
}
else
{
    // Add code to execute if a is less than or equal
    // to 50 here.
}
```

Оператор выбора switch

Синтаксис

```
switch ([expression to check])
{
    case [test1]:
        ...
        [exit case statement]
    case [test2]:
        ...
        [exit case statement]
    default:
        ...
        [exit case statement]
}
```

Пример

```
switch(a)
{
    case 0:
        // Executed if a is 0.
        break;
    case 1:
    case 2:
    case 3:
        // Executed if a is 1, 2, or 3.
        break;
    default:
        // Executed if a is any
        // other value.
        break;
}
```

Оператор выбора switch

```
int caseSwitch = 1;  
...  
switch (caseSwitch)  
{  
    case 1:  
        Console.WriteLine("Case 1");  
        break;  
    case 2:  
        Console.WriteLine("Case 2");  
        break;  
    case 7 - 4:  
        Console.WriteLine("Case 3");  
        break;  
    case 4:  
        while (true)  
        {  
            Console.WriteLine("Endless looping. . . .");  
        }  
    default:  
        Console.WriteLine("Default case");  
        break;  
}
```

Оператор выбора switch

Каждый блок кода в операторе switch должен заканчиваться оператором, который явно завершает конструкцию ([exit case statement]). Если опустить этот оператор, созникнет ошибка компиляции. В качестве таких операторов можно использовать:

```
switch(carColor.ToLower())
{
    case "red":
        // Red car
        break;
    case "blue":
        // Blue car
        break;
    default:
        // Unknown car
        break;
}
```

break;

goto case [testX];

return;

throw;

Рекомендации по использованию операторов выбора

Необходимо использовать конструкцию `if`, если существует одно условие, осуществляющее контроль за исполнением одного блока кода

Необходимо использовать конструкцию `if/else`, если существует одно условие, осуществляющее контроль за исполнением одного из двух блоков кода

Необходимо использовать конструкцию `if else if...` для запуска одного из нескольких блоков кода на основе условий, которые состоят из нескольких переменных

Необходимо использовать вложенные конструкции `if` при более сложном анализе условий, состоящих из нескольких переменных

Необходимо использовать оператор `switch` для выполнения действий, основанных на возможных значения одной переменной

Типы операторов цикла

В C# существует три типа операторов цикла

Цикл while

Цикл while позволяет выполнять блок кода ноль или более раз

Цикл do

Цикл работает аналогично циклу while, условие проверяется в конце итерации

Цикл for

Цикл for позволяет выполнять код заданное количество раз

Оператор while

Синтаксис

```
while ([condition])
{
    [Code to loop]
}
```

Пример

```
double balance = 100D;
double rate = 2.5;
double targetBalance = 1000D;
int years = 0;
while (balance <= targetBalance)
{
    balance *= (rate / 100) + 1;
    years += 1;
}
```

Цикл while содержит следующие элементы:

Ключевое слово while, которое определяет цикл while

Условие, которое проверяется в начале каждой итерации

Блок кода, который выполняется на каждой итерации

Оператор do...while

Синтаксис

```
do  
{  
    [Code to loop]  
} while ([condition]);
```

Пример

```
string userInput = "";  
do  
{  
    userInput = GetUserInput();  
    if (userInput.Length < 5)  
    {  
        // You must enter at least 5 characters.  
    }  
} while (userInput.Length < 5);
```

Синтаксис цикла do содержит следующие элементы:

Ключевое слово do, которое определяет цикл do

Блок кода, который выполняется на каждой итерации

Условие, которое проверяется в конце каждой итерации

Оператор for

Синтаксис

```
for ([counter variable] = [starting value]; [limit]; [counter modification])
{
    [Code to loop]
}
```

Синтаксис цикла for содержит следующие элементы:

Ключевое слово for для определения цикла for

Спецификацию цикла, состоящую из следующих элементов:

- переменная-счетчик
- начальное значение для переменной-счетчика
- предельное значение для переменной-счетчика
- инструкции о том, как изменять переменную-счетчик в конце каждой итерации

Блок кода для выполнения каждой итерации

Оператор for

```
for (int i = 0; i < 10; i++)
{
    // Code to loop, which can use i.
}

. . .

for (int i = 0; i < 10; i += 2)
{
    // Code to loop, which can use i.
}

. . .

int j;
for (j = 0; j < 10; j++)
{
    // Code to loop, which can use j.
}
// j is also available here
```

Оператор for

Можно использовать вложенные циклы for, каждый из которых определяет свою собственную переменную-счетчик

```
string[] strings = new string[]{"One", "Two", "Three", "Four", "Five"};
string result = "";
for(int stringIndex = 0; stringIndex < strings.Length; stringIndex++)
{
    for (int charIndex = strings[stringIndex].Length - 1; charIndex >= 0;
charIndex--)
    {
        result += strings[stringIndex][charIndex];
    }
}
```

result = ?

Операторы break и continue

Оператор break

```
int[] oldNumbers = { 1, 2, 3, 4, 5, 6, 7, 8 };
int count = 0;
while (oldNumbers.Length > count)
{
    if (oldNumbers[count] == 5)
    {
        break;
    }
    count++;
}
```

Оператор `break` позволяет полностью выйти из цикла и перейти к следующей строке кода вне цикла

Операторы break и continue

Оператор continue

```
int[] oldNumbers = { 1, 2, 3, 4, 5, 6, 7, 8 };
int count = 0;
while (oldNumbers.Length > count)
{
    if (oldNumbers[count] == 5)
    {
        continue;
    }
    count++;
}
```

Оператор `continue` похож на оператор `break` за исключением того, что, вместо выхода из цикла полностью, он пропускает оставшийся код текущей итерации, проверяет условие, и, в случае его истинности, начинает следующую итерацию цикла

Массивы в C#

Что такое массив?

Массив представляет собой набор объектов, которые сгруппированы вместе и управляются как единое целое

Массивы имеют следующие характеристики:

Каждый элемент в массиве содержит значение

Индексируются с нуля

Длина массива это общее число элементов, которое он может содержать

Нижняя граница массива индекс его первого элемента

Могут быть одномерными, многомерными или непрямоугольные

Ранг массива это число измерений в массиве

Создание и инициализация массивов

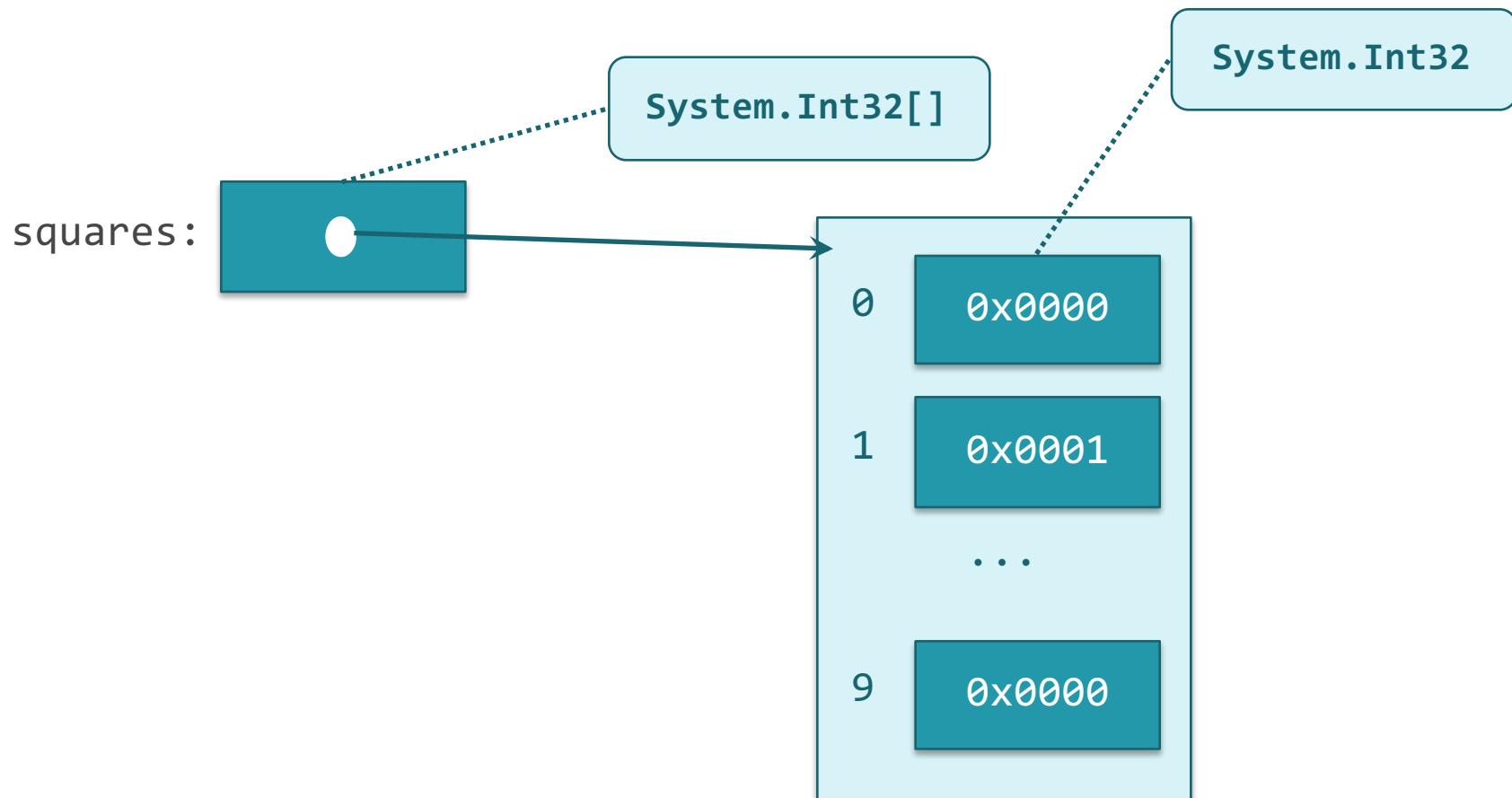
Одномерные (single, sz) массивы

```
int[] arrayName;  
. . .  
int[] list;  
list = new int[20];  
. . .  
int[] list = new int[20];  
. . .  
int[] list = new int[5] { 1, 2, 3, 4, 5 };  
int[] list = new int[] { 1, 2, 3, 4, 5 };  
int[] list = new[] { 1, 2, 3, 4, 5 };  
int[] list = { 1, 2, 3, 4, 5 };
```

Если не инициализировать элементы массива, компилятор C# инициализирует их автоматически при его создании с помощью ключевого слова new значениями по умолчанию для его базового типа

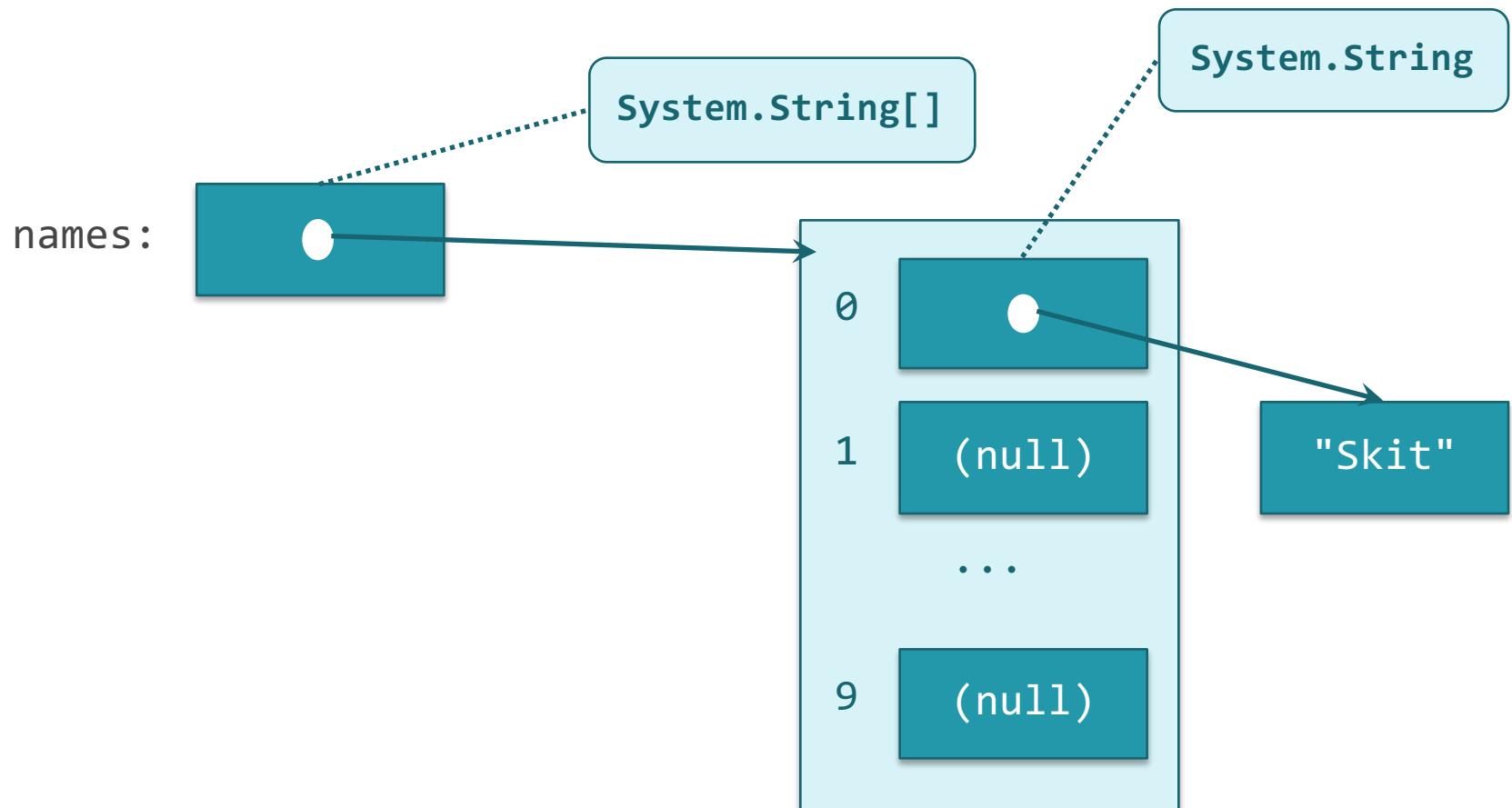
Создание и инициализация массивов

```
int[] squares = new int[10];  
squares[1] = 1;
```



Создание и инициализация массивов

```
string[] names = new string[10];  
names[0] = "Skit";
```



Создание и инициализация массивов

Многомерные (multiple) массивы

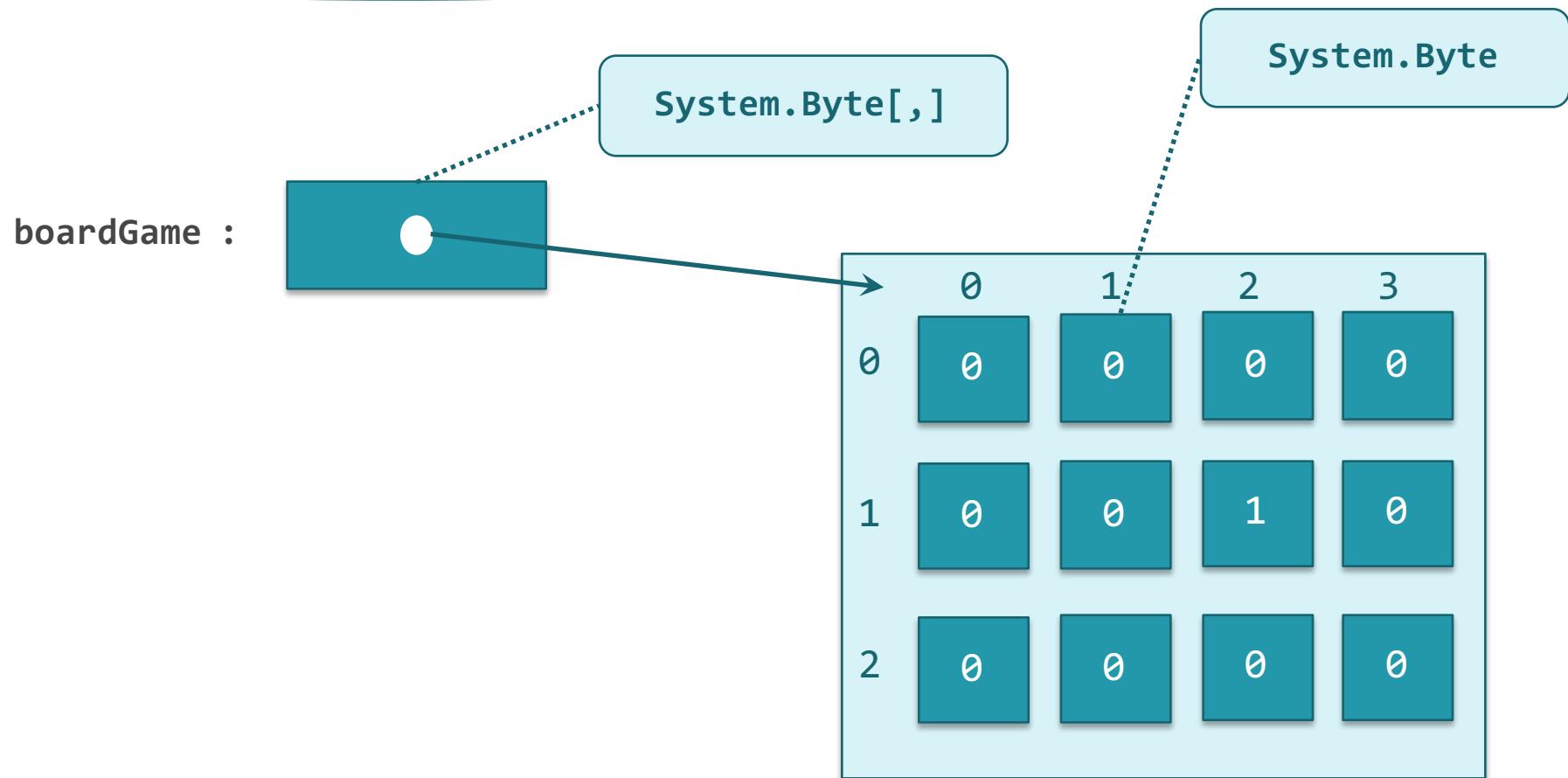
```
int[,] table; // two-dimensional array  
table = new int[10, 2];  
.  
.  
.  
int[, ,] cube = new int[3, 2, 5]; // three-dimensional array
```

Синтаксис

```
Type[ , , ... ] arrayName = new Type[ Size1, Size2 , . . . ];
```

Создание и инициализация массивов

```
byte[,] boardGame = new byte[3,4];  
boardGame[1,2] = 1;
```



Создание и инициализация массивов

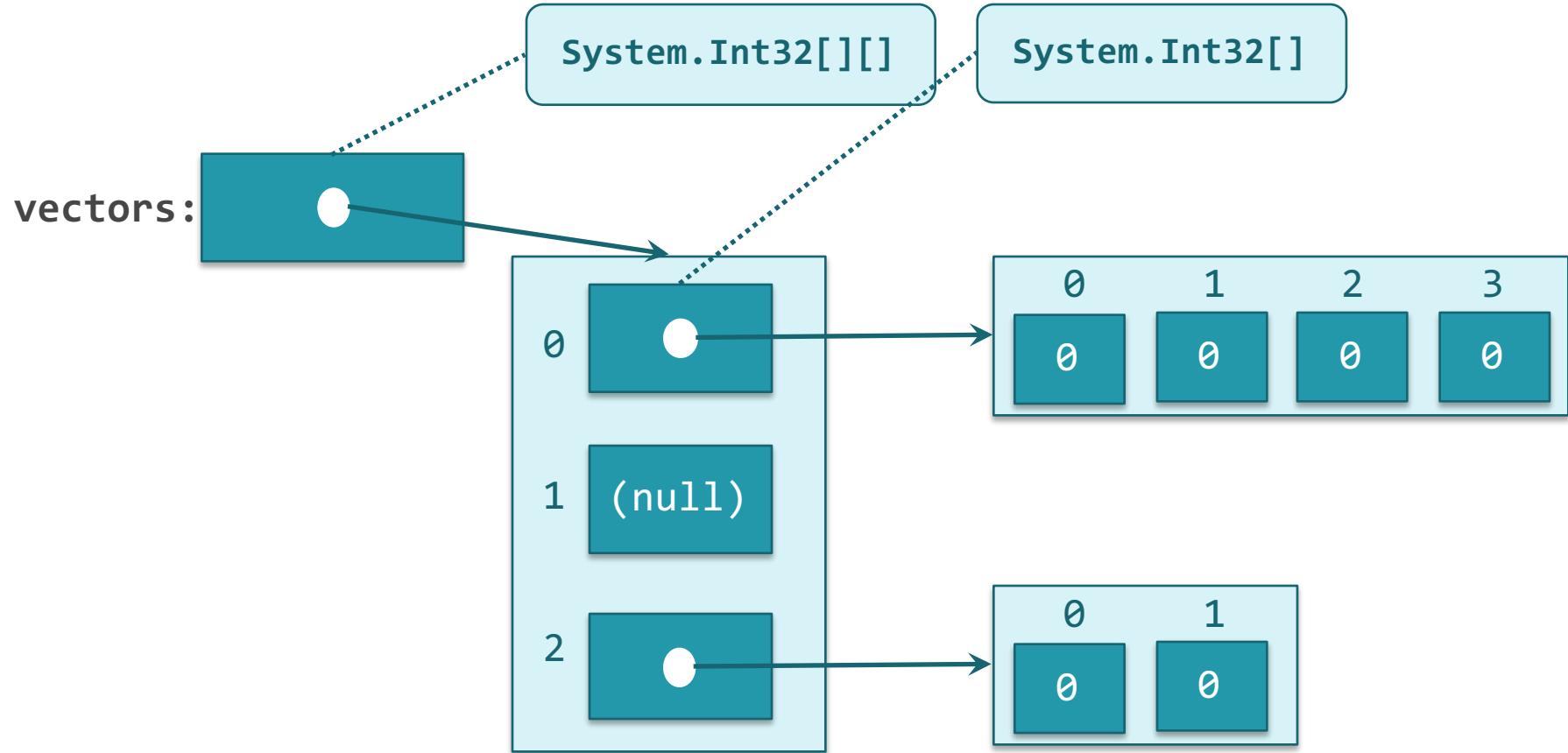
Массивы (jagged) массивов

```
Type [][] jaggedArray = new Type[10][];  
jaggedArray[0] = new Type[5]; // Can specify different sizes  
jaggedArray[1] = new Type[7];  
...  
JaggedArray[9] = new Type[21];
```

```
int[][][] jaggedArray = new int[3][,]  
{  
    new int[,] {{1, 3}, {5, 7}},  
    new int[,] {{0, 2}, {4, 6}, {8, 10}},  
    new int[,] {{11, 22}, {99, 88}, {0, 9}}  
};
```

Создание и инициализация массивов

```
int[,] vectors = new int[3][];  
vectors[0] = new int[4];  
vectors[2] = new int[2];
```



Создание и инициализация массивов

Неявно типизированные массивы

```
var mixed = new[] { 1, DateTime.Now, true, false, 1.2 };
```



CTE

```
// int[]
var a = new[] { 1, 10, 100, 1000 };
// string[]
var b = new[] { "hello", null, "world" };
```

```
// jagged array of strings
var d = new[]
{
    new[] {"Luca", "Mads", "Luke", "Dinesh"},
    new[] {"Karen", "Suma", "Frances"}
};
```

```
// single-dimension jagged array
var c = new[]
{
    new[] {1, 2, 3, 4},
    new[] {5, 6, 7, 8}
};
```

Создание и инициализация массивов

Приведение типов в массивах

```
string[] strings = new string[3]{"one", "two", "three"};
object[] objects = new object[3];
objects = strings;
string[] stringsAgain = new string[3];
stringsAgain = (string[])objects;
```

CLR не допускает приведение массивов с элементами значимых типов к другому типу

```
int[] integers = new int[3] { 1, 2, 3 };
object[] objects = new object[3];
objects = integers; //CTE
```



```
int[] integers = new int[5];
object[] objects = new object[integers.Length];
Array.Copy(integers, objects, integers.Length);
```

Передача и возврат массивов

Массив передается в метод всегда по ссылке, а метод может модифицировать элементы в массиве

Отдельные методы могут возвращать ссылку на массив

Если метод создает и инициализирует массив, возвращение ссылки на массив не вызывает проблем; если же нужно, чтобы метод возвращал ссылку на внутренний массив, ассоциированный с полем, сначала необходимо решить, вправе ли вызывающая программа иметь доступ к этому массиву

Символы, строки и работа с текстом

Символы

Метод	Действие
GetUnicodeCategory	Метод возвращает элементы перечисления <code>UnicodeCategory</code> , описывающего категорию символа
IsDigit	Возвращает <code>true</code> , если символ является десятичной цифрой
IsLetter	Возвращает <code>true</code> , если символ является буквой
IsPunctuation	Возвращает <code>true</code> , если символ является знаком препинания
IsSeparator	Возвращает <code>true</code> , если символ является разделителем
IsLower	Возвращает <code>true</code> , если символ – это буква в нижнем регистре
IsUpper	Возвращает <code>true</code> , если символ – это буква в верхнем регистре
ToLower	Приводит символ к нижнему регистру
ToUpper	Приводит символ к верхнему регистру
IsControl	Возвращает <code>true</code> , если символ является управляемым
IsLetterOrDigit	Возвращает <code>true</code> , если символ является буквой или цифрой
IsNumber	Возвращает <code>true</code> , если символ является десятичной или шестнадцатеричной цифрой

Символы

```
double d = Char.GetNumericValue('\u0033');
Console.WriteLine(d.ToString());//3

// '\u00bc' is the “vulgar fraction one quarter ('¼'’)
d = Char.GetNumericValue('\u00bc');
Console.WriteLine(d.ToString());//0.25

// 'A' is the “Latin capital letter A”
d = Char.GetNumericValue('A');
Console.WriteLine(d.ToString());//-1
```

```
char c;
int n;
c = (char)65;
Console.WriteLine(c); //A
n = (int)c;
Console.WriteLine(n); //65
c = Convert.ToChar(65);
Console.WriteLine(c); //A
n = Convert.ToInt32(c);
Console.WriteLine(n); //65
```

Строки

```
string s = new string("Hi there.");  
String s = new String("Hi there.");
```

СТЕ

```
string s = "Hi there.";  
String s = "Hi there.";
```

У класса String достаточно много конструкторов, позволяющих задать строку

```
string s = new string(' ', 20);
```

```
char[] a = { 'a', 'b', 'c', 'd', 'e' };  
string s = new string(a);
```

```
string s = "Hi\r\nthere.";  
string s = "Hi" + Environment.NewLine + "there.";
```

```
string file = "C:\\Windows\\System32\\Notepad.exe";
```

```
string file = @"C:\\Windows\\System32\\Notepad.exe"; //verbatim strings
```

Строки

Член	Описание
Compare (статический метод)	Сравнение двух строк в лексикографическом (алфавитном) порядке. Разные реализации метода позволяют сравнивать строки с учетом, или без учета регистра
CompareTo (экземплярный метод)	Сравнение текущего экземпляра строки с другой строкой
Concat (статический метод)	Слияние произвольного числа строк
Copy (статический метод)	Создание копии строки
Empty (статическое поле)	Открытое статическое поле, представляющее пустую строку
Format (статический метод)	Форматирование строки в соответствии с заданным форматом

Строки

Член	Описание
IndexOf, LastIndexOf (экземплярные методы)	Определение индекса первого или, соответственно, последнего вхождения подстроки в данной строке
IndexOfAny, LastIndexOfAny (экземплярные методы)	Определение индекса первого или, соответственно, последнего вхождения любого символа из подстроки в данной строке
Insert (экземплярный методы)	Вставка подстроки в заданную позицию
Join (статический метод)	Слияние массива строк в единую строку. Между элементами массива вставляются разделители
Length (свойство)	Возвращает длину строки

Строки

Член	Описание
PadLeft, PadRigth (экземплярные методы)	Выравнивают строки по левому или, соответственно, правому краю путем вставки нужного числа пробелов в начале, или в конце строки
Remove (экземплярный метод)	Удаление подстроки из заданной позиции
Replace (экземплярный метод)	Замена всех вхождений заданной подстроки, или символа новыми подстрокой, или символом
Split (экземплярный метод)	Разделяет строку на элементы, используя разные разделители. Результаты помещаются в массив строк
EndWith EndWith (экземплярные методы)	Возвращают true или false в зависимости от того, начинается, или заканчивается строка заданной подстрокой

Строки

Член	Описание
Substring (экземплярный метод)	Выделение подстроки, начиная с заданной позиции
ToCharArray (экземплярный метод)	Преобразует строку в массив символов
ToLower, ToUpper (экземплярные методы)	Преобразование строки к нижнему или, соответственно, к верхнему регистру
Trim, TrimStart, TrimEnd (экземплярные методы)	Удаление пробелов в начале и конце строки, или только с начала, или только с конца соответственно

Строки

```
string s = "Это текстовая строка, состоящая из семи слов.";  
  
Console.WriteLine("Длина строки {0}", s.Length);  
Console.WriteLine("Наличие подстроки \"текст\"? {0} ", s.Contains("текст"));  
Console.WriteLine("Вставка \"{0}\"", s.Insert(4, "большая "));  
Console.WriteLine("Длина строки {0} символов", s.Length);  
Console.WriteLine("Удаление: \"{0}\"", s.Remove(4, 10));  
Console.WriteLine("Замена: \"{0}\"", s.Replace("семи", "нескольких"));  
Console.WriteLine("Подстрока: \"{0}\"", s.Substring(4, 5));  
Console.WriteLine("В верхний регистр: {0}", s.ToUpper());  
Console.WriteLine("Вхождение \"текст\": {0}", s.IndexOf("текст"));
```

Строки

```
string s = "a";
```

```
Do(s);
```

```
·     void Do(string p)  
·     {  
·         p += "b";  
·     }
```

```
Console.WriteLine(s);
```

"a"

System.String

"ab"

1

2

3

4

Строки

Пример использования методов разделения строки на элементы Split и слияние массива строк в единую строку Join

```
string source = "тучки небесные вечные странники";
char[] separators = { ' ' };
string[] parts = source.Split(separators);
Console.WriteLine("Результат разбиения строки на части: ");
for (int i = 0; i < parts.Length; i++)
{
    Console.WriteLine(parts[i]);
}
string result = String.Join(" | ", parts);
Console.WriteLine("Результат сборки: ");
Console.WriteLine(result);
```

Тип `string` является неизменяемым (`immutable`) типом данных, таким образом, методы и операции, модифицирующие содержимое строк, на самом деле создают новые строки, копируя при необходимости содержимое старых

Строки

Для того, чтобы разрабатываемые классы были дружественными к пользователю, они должны предлагать средства для отображения своих строковых представлений в любом из форматов, которые могут понадобиться пользователю

В исполняющей среде .NET определен стандартный способ достижения этого — интерфейс `IFormattable`

Применение форматной строки к примитивному типу

```
decimal d = 12.05667M;  
int i = 5;  
Console.WriteLine("Значение переменной d = {0:c}, а i = {1}", d, i);
```

Указывается индекс

Может быть включена и другая информация, относящаяся к формату данного элемента

- ✓ количество символов, которое займет представление элемента, снабженное префиксом-запятой
- ✓ спецификатор формата предваряется двоеточием

Строки

Спецификатор	Применяется к	Значение	Пример
C	Числовым типам	Символ местной валюты	\$835.50 (США) £835.50 (Великобритания) 835.50р.(Россия)
D	Только к целочисленным типам	Обычное целое	835
E	Числовым типам	Экспоненциальная нотация	8.35E+002
F	Числовым типам	С фиксированной десятичной точкой	835.50
G	Числовым типам	Обычные числа	835.5
N	Числовым типам	Формат чисел, принятый в данной местности	4,384.50 (Великобритания/США) 4 384,50 (континентальная Европа)
P	Числовым типам	Процентная нотация	835,000.00%
X	Только к целочисленным типам	Шестнадцатеричный формат	1a1f

Строки

```
CultureInfo ci = new CultureInfo("en-us");
double floating = 10761.937554;
Console.WriteLine("C: {0}", floating.ToString("C", ci));
Console.WriteLine("E: {0}", floating.ToString("E03", ci));
Console.WriteLine("F: {0}", floating.ToString("F04", ci));
Console.WriteLine("G: {0}", floating.ToString("G", ci));
Console.WriteLine("N: {0}", floating.ToString("N03", ci));
Console.WriteLine("P: {0}", (floating / 10000).ToString("P02", ci));
Console.WriteLine("R: {0}", floating.ToString("R", ci));
Console.WriteLine();
```

C: \$10,761.94
E: 1.076E+004
F: 10761.9376
G: 10761.937554
N: 10,761.938
P: 107.62 %
R: 10761.937554

```
CultureInfo ci = new CultureInfo("ru-ru");
double floating = 10761.937554;
Console.WriteLine("C: {0}", floating.ToString("C", ci));
Console.WriteLine("E: {0}", floating.ToString("E03", ci));
Console.WriteLine("F: {0}", floating.ToString("F04", ci));
Console.WriteLine("G: {0}", floating.ToString("G", ci));
Console.WriteLine("N: {0}", floating.ToString("N03", ci));
Console.WriteLine("P: {0}", (floating / 10000).ToString("P02", ci));
Console.WriteLine("R: {0}", floating.ToString("R", ci));
Console.WriteLine();
```

C: 10 761,94p.
E: 1,076E+004
F: 10761,9376
G: 10761,937554
N: 10 761,938
P: 107,62%
R: 10761,937554

Строки

```
CultureInfo ci = new CultureInfo("ru-ru");
int integral = 8395;
Console.WriteLine("C: {0}", integral.ToString("C", ci));
Console.WriteLine("D: {0}", integral.ToString("D6", ci));
Console.WriteLine("E: {0}", integral.ToString("E03", ci));
Console.WriteLine("F: {0}", integral.ToString("F01", ci));
Console.WriteLine("G: {0}", integral.ToString("G", ci));
Console.WriteLine("N: {0}", integral.ToString("N01", ci));
Console.WriteLine("P: {0}", (integral / 10000).ToString("P02", ci));
Console.WriteLine("X: 0x{0}", integral.ToString("X", ci));
```

C: 8 395,00p.
D: 008395
E: 8,395E+003
F: 8395,0
G: 8395
N: 8 395,0
P: 0,00%
X: 0x20CB

```
CultureInfo ci = new CultureInfo("en-us");
int integral = 8395;
Console.WriteLine("C: {0}", integral.ToString("C", ci));
Console.WriteLine("D: {0}", integral.ToString("D6", ci));
Console.WriteLine("E: {0}", integral.ToString("E03", ci));
Console.WriteLine("F: {0}", integral.ToString("F01", ci));
Console.WriteLine("G: {0}", integral.ToString("G", ci));
Console.WriteLine("N: {0}", integral.ToString("N01", ci));
Console.WriteLine("P: {0}", (integral / 10000).ToString("P02", ci));
Console.WriteLine("X: 0x{0}", integral.ToString("X", ci));
```

C: \$8,395.00
D: 008395
E: 8.395E+003
F: 8395.0
G: 8395
N: 8,395.0
P: 0.00 %
X: 0x20CB

Строковый тип `StringBuilder`

Для динамических операций со строками и символами в библиотеке классов .NET Framework существует тип `TextStringBuilder` пространства имен `System.Text`

```
StringBuilder sb = new StringBuilder();
```

Каждый конструктор `StringBuilder` обязан выделять память и инициализировать три внутренних поля, управляемых любым объектом `StringBuilder`.

- Максимальная емкость (Maximum capacity) – поле типа `int`, задающее максимальное число символов, размещаемых в строке
- Емкость (Capacity) – поле типа `int`, показывающее размер массива символов `StringBuilder`
- Массив символов (Character array) – массив структур `char`, содержащий набор символов строки. Число символов всегда меньше (или равно) емкости и максимальной емкости

Строковый тип `StringBuilder`

```
public sealed class StringBuilder : ISerializable // .NET 2.0
{
    // Fields
    private const string CapacityField = "Capacity";
    internal const int DefaultCapacity = 0x10;
    internal IntPtr m_currentThread;
    internal int m_MaxCapacity;
    internal volatile string m_StringValue; // HERE -----
    private const string MaxCapacityField = "m_MaxCapacity";
    private const string StringValueField = "m_StringValue";
    private const string ThreadIDField = "m_currentThread";
    . . .
}
```

Строковый тип `StringBuilder`

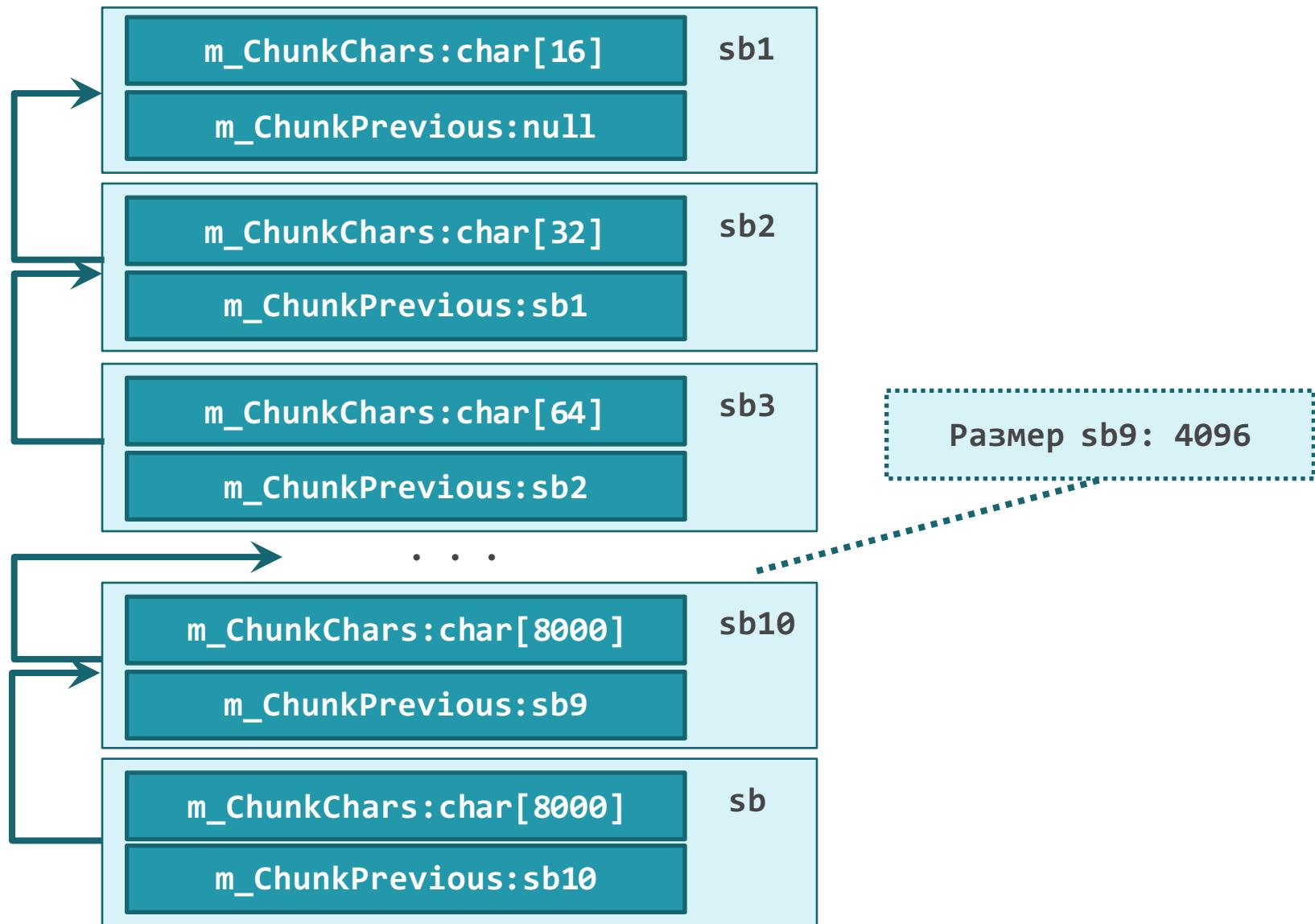
```
public sealed class StringBuilder : ISerializable // .NET 4.0
{
    // Fields
    private const string CapacityField = "Capacity";
    internal const int DefaultCapacity = 0x10;
    internal char[] m_ChunkChars; // HERE -----
    internal int m_ChunkLength;
    internal int m_ChunkOffset;
    internal StringBuilder m_ChunkPrevious;
    internal int m_MaxCapacity;
    private const string MaxCapacityField = "m_MaxCapacity";
    internal const int MaxChunkSize = 0x1f40; //8000
    private const string StringValueField = "m_StringValue";
    private const string ThreadIDField = "m_currentThread";
    . . .
}
```

Строковый тип StringBuilder

```
static void Main(string[] args)
{
    var sb = new StringBuilder();
    Append(sb, 20000);
    sb.Clear();
}

private static void Append(StringBuilder sb, int number0fChars)
{
    for (int i = 0; i < number0fChars; i++)
    {
        sb.Append("F");
    }
}
```

Строковый тип StringBuilder



Строковый тип `StringBuilder`

Член	Описание
MaxCapacity (свойство)	Возвращает наибольшее количество символов
Capacity (свойство)	Получает/устанавливает размер массива символов
EnsureCapacity (метод)	Гарантирует, что размер массива символов будет не меньше, чем значение параметра, передаваемого этому методу
Length (свойство)	Возвращает количество символов в строке
ToString (метод)	Версия без параметров возвращает объект <code>String</code> , представляющий поле с массивом символов объекта <code>StringBuilder</code>
AppendFormat (метод)	Добавляет заданные объекты в массив символов, увеличивая его при необходимости

Строковый тип `StringBuilder`

Член	Описание
Replace (метод)	Заменяет один символ или строку символов в массиве символов
Remove (метод)	Удаляет диапазон символов из массива символов
Equals (метод)	Возвращает <code>true</code> , только если объекты <code>StringBuilder</code> имеют одну и ту же максимальную емкость, емкость и одинаковые символы в массиве
CopyTo (метод)	Копирует подмножество символов <code>StringBuilder</code> в массив <code>Char</code>

Строковый тип **StringBuilder**

```
StringBuilder sb = new StringBuilder();
sb.AppendFormat("{0} {1}", ".NET", "---Framework");
string s = sb.Replace("-", " ").Remove(4, 3).ToString();
```

```
StringBuilder sb = new StringBuilder();
sb.AppendFormat("{0}{1}", ".NET", "Framework");
string s = sb.ToString().ToUpper();
sb.Length = 0;
sb.Append(s).Insert(4, " ");
s = sb.ToString();
Console.WriteLine(s); // .NET FRAMEWORK
```

Регулярные выражения

Для упрощения решения задач по обработке символьной информации в пространстве имен `System.Text.RegularExpressions` определены классы для работы со строками, основанные на регулярных выражениях, `Regex`, `Match` и `MatchCollection`

Регулярное выражение – это шаблон, согласно которому выполняется поиск соответствующего фрагмента текста

Использование регулярных выражений обеспечивает:

- проверку строки на соответствие шаблону
 - поиск в тексте по заданному шаблону
 - разбиение текста на фрагменты
-
- Набор управляющих кодов для идентификации специфических типов символов
 - Система для группирования частей подстрок и промежуточных результатов таких действий

Регулярные выражения

Набор символов	Описание	Пример
.	Любой символ, кроме \n	Выражение c.t соответствует фрагментам: cat, cut, c#t, c{t и т.д.
[]	Любой одиночный символ из записанной внутри скобок. Допускается использование диапазонов символов, для задания которого используется символ «-»	Выражение c[au]t соответствует фрагментам: cat, cut, cit. Выражение c[a-c]t соответствует фрагментам: cat, cbt, cct
[^]	Любой одиночный символ, не входящий в последовательность, записанную внутри скобок. Допускается использование диапазонов символов	Выражение c[^au]t соответствует фрагментам: cbt, cct, c2t и т.д. Выражение c[^a-c]t соответствует фрагментам: cdt, cet, c%t и т.д.

Регулярные выражения

Набор символов	Описание	Пример
\w	Любой алфавитно-цифровой символ, а также символ подчеркивания	Выражение <code>c\wt</code> соответствует фрагментам: <code>cbt</code> , <code>cct</code> , <code>c2t</code> , <code>c_t</code> и т. д., но не соответствует фрагментам <code>c%t</code> , <code>c{t</code> и т. д.
\W	Любой символ не удовлетворяющий \w	Выражение <code>c\Wt</code> соответствует фрагментам: <code>c%t</code> , <code>c{t</code> , <code>c.t</code> и т.д., но не соответствует фрагментам <code>cbt</code> , <code>cct</code> , <code>c2t</code> и т.д.
\s	Любой пробельный символ (пробел, табуляция и переход на новую строку)	Выражение <code>\s\w\w\w\s</code> соответствует любому слову из трех букв, окруженному пробельными символами

Регулярные выражения

Набор символов	Описание	Пример
\s	Любой не пробельный символ	Выражение <code>\s\S\S\s</code> соответствует любым трем непробельным символам, окруженным пробельными
\b	Граница слова	Выражение <code>\B\d\d\d\B</code> соответствует любым трем цифрам, входящим в состав слова так, что ни справа ни слева от них нет конца слова
\d	Любая десятичная цифра	Выражение <code>c\dt</code> соответствует фрагментам: c1t, c2t, c3t и т.д.

Регулярные выражения

Повторители	Описание	Пример
*	Ноль, или более повторений предыдущего элемента	Выражение ca^*t соответствует фрагментам: ct, cat, caat, caaat и т.д.
+	Одно, или более повторений предыдущего элемента	Выражение $ca+t$ соответствует фрагментам: cat, caat, caaat и т.д.
?	Не более одного повторения предыдущего элемента	Выражение $ca?t$ соответствует фрагментам: ct, cat
{n}	Ровно n повторений предыдущего элемента	Выражение $ca\{3\}t$ соответствует фрагменту: caaat. Выражение $(cat)\{2\}$ соответствует фрагменту: catcat

Регулярные выражения

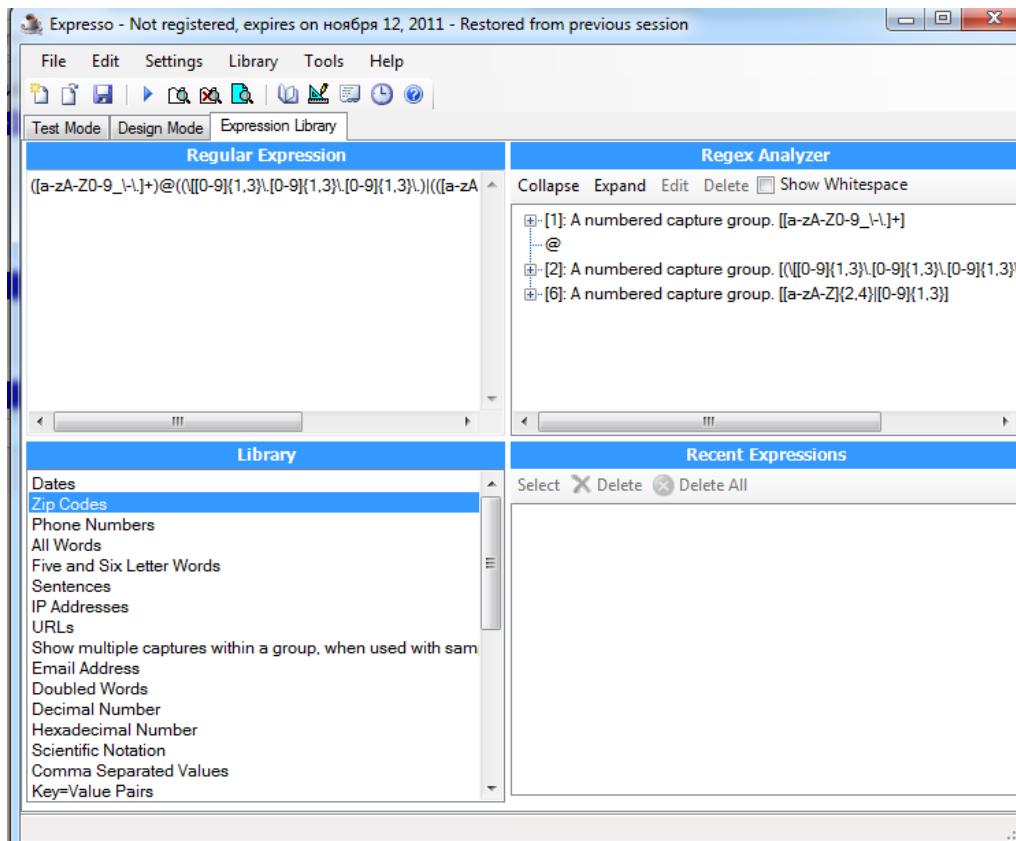
Повторители	Описание	Пример
{n,}	По крайней мере n повторений предыдущего элемента	Выражение соответствует фрагментам: caaat, caaaat, caaaaaat и т.д. Выражение соответствует фрагментам: catcat, catcatcat и т.д.
{n, m}	От n до m повторений предыдущего элемента	Выражение соответствует фрагментам: saat, caaat, caaaaat

Регулярные выражения

- Слово USA – @ " USA"
- Номер телефона в формате xxx-xx-xx – @"\d\d\d-\d\d-\d\d" или @"\d{3}(-\d\d){2}"
- Целое число со знаком, или без знака – @"[+-]?\d+"
- Время в формате чч.мм, или чч:мм – @"([01]\d)|(2[0-4])[\.:][0-5]\d"

Регулярные выражения

Редактор Expresso подходит в качестве учебного пособия для начинающего пользователя регулярных выражений, а также является полнофункциональной средой разработки для опытного программиста или веб-дизайнер с обширными знаниями о регулярных выражений



Регулярные выражения

Класс Regex

- IsMatch
- Match
- Matches
- Replace
- Split

```
Regex r = new Regex("собака",
RegexOptions.IgnoreCase);
string text1 = "Кот в доме, собака в конуре.";
string text2 = "Котик в доме, собачка в конуре.";
Console.WriteLine(r.IsMatch(text1)); //True
Console.WriteLine(r.IsMatch(text2)); //False
```

```
Regex r = new Regex(@"\d{2,3}(-\d\d){2}");
string text1 = "tel:123-45-67";
string text2 = "tel:no";
string text3 = "tel:12-34-56";
Console.WriteLine(r.IsMatch(text1));//True
Console.WriteLine(r.IsMatch(text2));//False
Console.WriteLine(r.IsMatch(text3));//True
```

Регулярные выражения

Класс Match

- Success
- Length
- Index
- NextMatch

```
Regex r = new Regex(@"[-+]?[0-9]+");
string text = @"5*10=50 -80/40=-2";
Match teg = r.Match(text);
int sum = 0;
while (teg.Success)
{
    Console.WriteLine("{0} ", teg.Value);
    sum += int.Parse(teg.ToString());
    teg = teg.NextMatch();
}
Console.WriteLine("\nsum={0}", sum);
```

Регулярные выражения

Класс MatchCollection

Метод Matches класса Regex возвращает объект класса MatchCollection – коллекцию только для чтения всех фрагментов заданной строки, совпадавших с шаблоном

```
Regex r = new Regex(@"[-+]?[0-9]+");
string text = @"5*10=50 -80/40=-2";
MatchCollection teg = r.Matches(text);
int sum = 0;
foreach (Match temp in teg)
{
    sum += int.Parse(temp.ToString());
}
Console.WriteLine("\nsum={0}", sum);
```

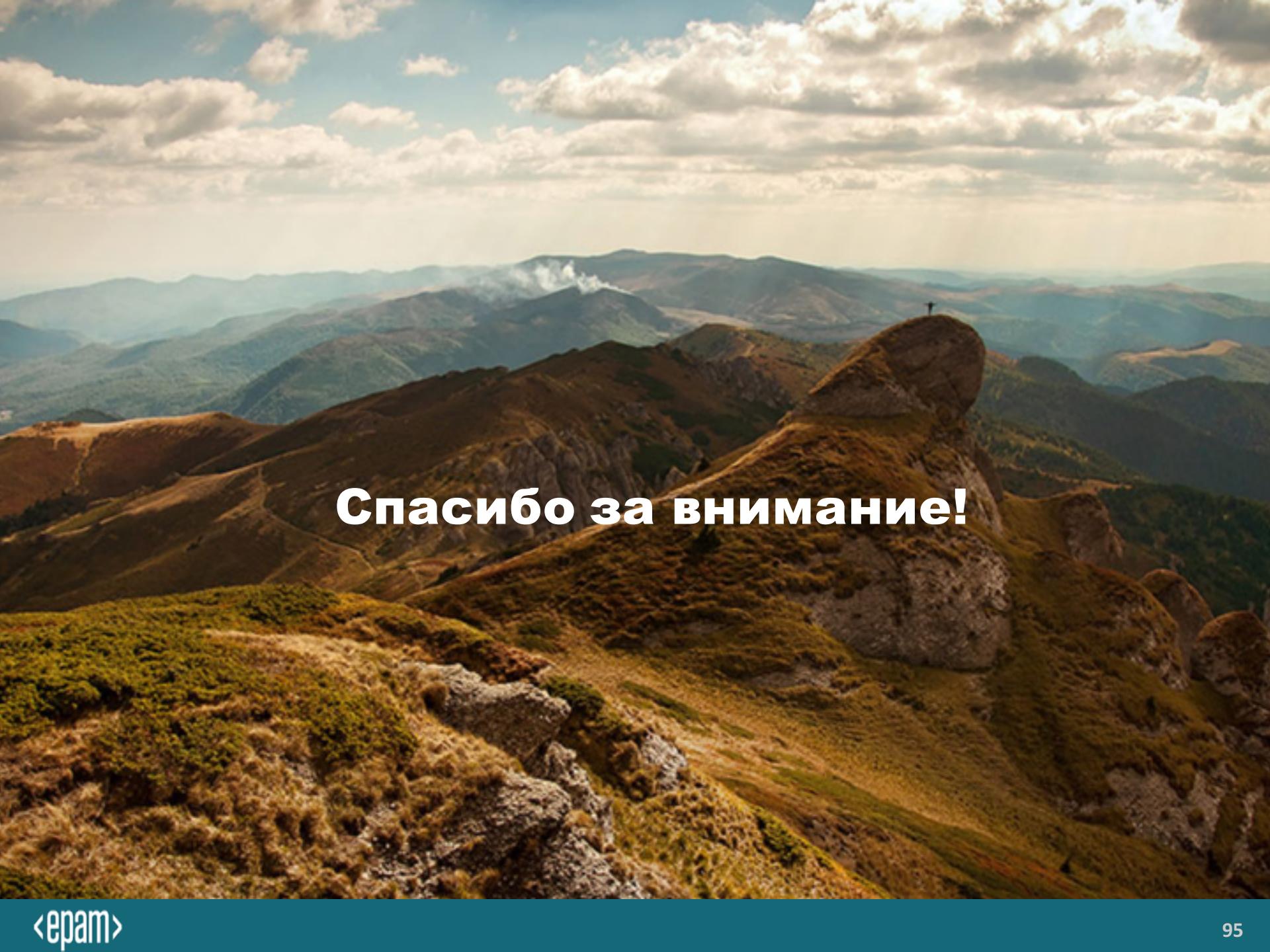
Регулярные выражения

Статический метод Replace

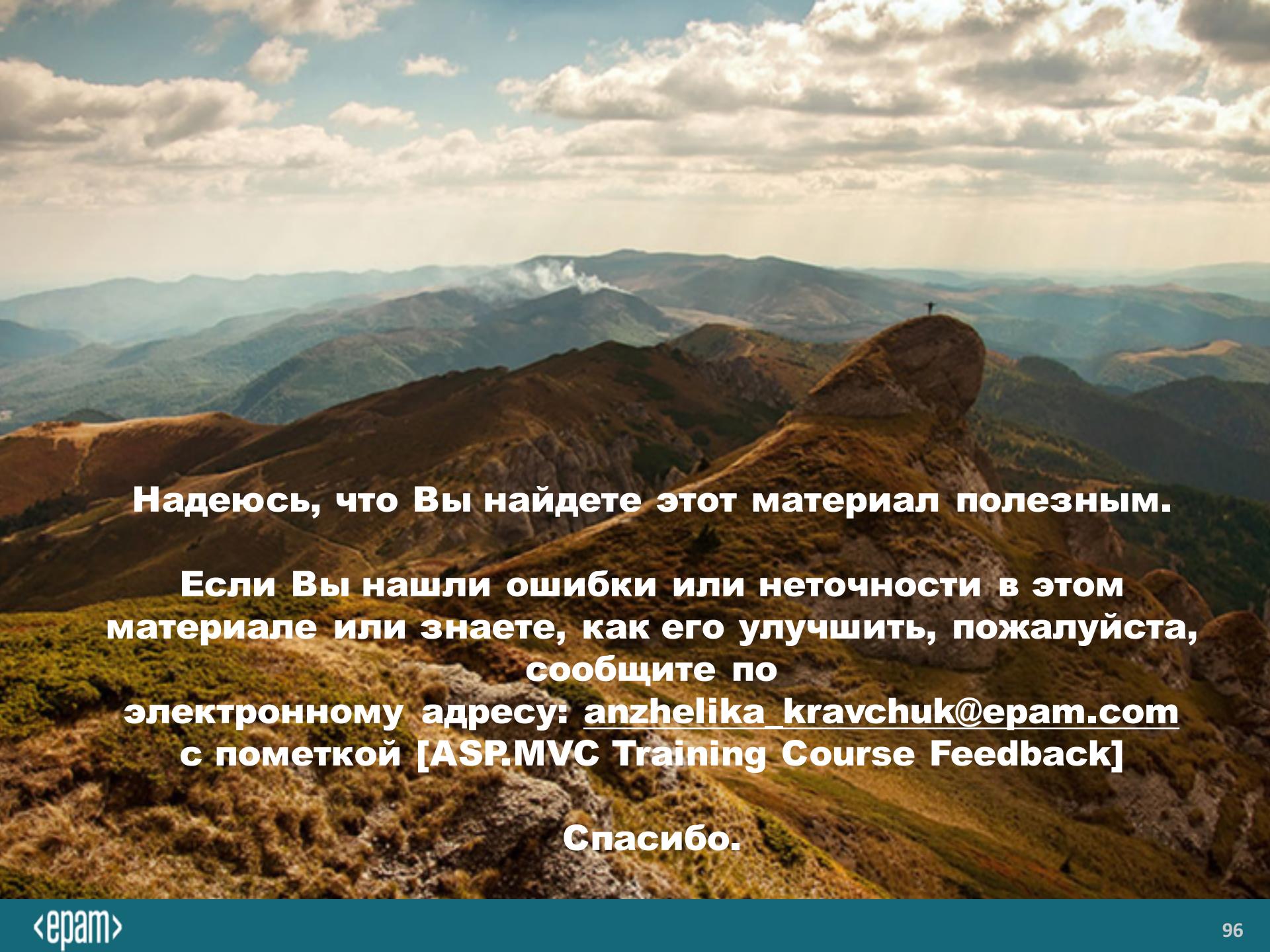
```
string text = @"Контактные телефоны tel: 123-45-67, 123-34-56; fax:123-56-45";
Console.WriteLine("Старые данные\n" + text);
string newText = Regex.Replace(text, @"\d{2,3}(-\d\d){2}", "");
Console.WriteLine("Новые данные\n" + newText);
```

Статический метод Split

```
string text = @"Контакты телефоны: 123-45-67, 123-34-56";
string[] newText = Regex.Split(text, "[ ,.:;]+");
```

A wide-angle photograph of a mountain range under a dramatic sky. In the foreground, rocky terrain and green slopes are visible. A lone figure stands on a prominent peak in the middle ground. The background features multiple layers of mountains, with smoke or steam rising from one of the peaks in the distance.

Спасибо за внимание!

A wide-angle photograph of a mountainous landscape under a dramatic sky. In the foreground, a rocky mountain ridge slopes down towards the viewer. On the right side of the ridge, a small figure of a person stands on a prominent rock, looking out over the vast, rolling hills and mountains in the distance. The sky is filled with large, white, billowing clouds against a bright blue background.

Надеюсь, что Вы найдете этот материал полезным.

**Если Вы нашли ошибки или неточности в этом
материале или знаете, как его улучшить, пожалуйста,
сообщите по**

**электронному адресу: anzhelika_kravchuk@epam.com
с пометкой [ASP.MVC Training Course Feedback]**

Спасибо.