# КОНСТРУКЦИИ ЯЗЫКА, МАССИВЫ, СПИСКИ И СЛОВАРЬ

## C# - ЛЕКЦИЯ 4

# НА ПРОШЛОМ ЗАНЯТИИ

- Типы данных: классы, структуры, перечисления, интерфейсы, делегаты
- Члены типов
- Встроенные типы в CLI

# СЕГОДНЯ В ПРОГРАММЕ

- Конструкции языка
- Контроль переполнения: checked/uncheked
- Массивы
- Список
- Словарь

# ОБЪЯВЛЕНИЕ ПЕРЕМЕННЫХ VAR

```
var stopwatch = new Stopwatch();

Stopwatch stopwatch2 = new Stopwatch();
```

- Это не динамический тип
- Тип по-прежнему не может измениться
- Это лишь для красоты

# СОЗДАНИЕ ЭКЗЕМПЛЯРОВ

```
//  struct
DateTime dateTime = new DateTime();

//  class
Stopwatch stopwatch = new Stopwatch();

//  delegate
Func<double, double> sin = new Func<double, double>(Math.Sin);
```

```
//  struct
int i = 17;

//  class
string name = "John";

//  delegate
Func<double, double> cos = Math.Cos;
```

# NULLABLE<T>

```
Nullable<DateTime> birthDate = null;
```

```
DateTime? birthDate = null;
```

```
DateTime? birthDate = null;

if (birthDate.HasValue)
{
}
```

```
DateTime? birthDate = null;

if (birthDate != null)
{
}
```

```
DateTime? birthDate = null;

if (birthDate.HasValue)
{
    int age = DateTime.Now - birthDate.Value;
    //  ...
}
```

# ПРИВЕДЕНИЕ ТИПА

- Выражение приведения типа - для всех типов
- Оператор as - для ссылочных типов

# ВЫРАЖЕНИЕ ПРИВЕДЕНИЯ ТИПА

```
int i = (int)Math.Sin(3.44);
```

# ОПЕРАТОР AS

```
void Move(Animal a)
{
    Bird b = a as Bird;

    if (b != null)
        b.Fly();

    Dog d = a as Dog;

    if (d != null)
        d.Run();
}
```

# ПРОВЕРКА ТИПА

- Оператор is
- Метод GetType() и оператор typeof

# ОПЕРАТОР IS

```
void Move(Animal a)
{
    if (a is Bird)
        ((Bird)a).Fly();

    if (a is Dog)
        ((Dog)a).Run();
}
```

# МЕТОД GETTYPE() И ОПЕРАТОР TYPEOF

```
void Move(Animal a)
{
    if (a.GetType() == typeof(Bird))
        ((Bird)a).Fly();

    if (a.GetType() == typeof(Dog))
        ((Dog)a).Run();
}
```

# ТЕРНАРНЫЙ ОПЕРАТОР

```
Console.WriteLine("Введите значение или нажмите <Enter> для считывания его

string input = Console.ReadLine();
int value = input.Length > 0
    ? int.Parse(input)
    : ReadValueFromConfig();
```

# ОПЕРАТОР ОБЪЕДИНЕНИЯ С NULL

(null-coalescing operator)

```
string address = FindUserAddressInDb(login) ?? "Адрес не указан";
```

# ОПЕРАТОР ПРИСВАИВАНИЯ С ОБЪЕДИНЕНИЕМ С NULL

(null-coalescing assignment operator)

```
decimal GetRate(string from, string to)
{
    decimal? rate = TryFetchRate(from, to);
    rate ??= FallbackToLastSavedValue(from, to);
    rate ??= FallbackToConfig(from, to);

    return rate.Value;
}
```

# ДОСТУП К ЧЛЕНАМ ТИПА

- `.` - поля, методы, свойства, события
- `[ ]` - индексаторы

```
DateTime now = DateTime.Now;
DateTime nextWeek = now.AddDays(7);

string str = "Hello world";
char w = str[6];
```

## Защита от null

- `?.` - поля, методы, свойства, события
- `?[]` - индексаторы

```
var data = FetchPersonData();
int? birthYear = data.BirthDate?.Year;
var firstFriend = data.Friends?[0];
```

# Синтаксический сахар

```
var data = FetchPersonData();
int? birthYear;

if (data.BirthDate != null)
    birthYear = data.BirthDate.Year;

Person firstFriend;
if (data.Friends != null)
    firstFriend = data.Friends[0];
```

# Цепочки обращений с проверками

```
A?.B?.Do(C);
A?.B?[C];
```

# КОНТРОЛЬ ЗА ПЕРЕПОЛНЕНИЕМ ПРИ ВЫПОЛНЕНИИ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ

## ПЕРЕПОЛНЕНИЕ

Отбрасывание битов высокого порядка при выполнении операции, результат которой не умещается в тип результата.

# КОГДА ВОЗНИКАЕТ

- При арифметических операциях над целыми типами: `++`, `--`, `унарном -`, `+`, `-`, `*`, `/`
- При явном преобразовании типа между целыми типами
- При явном преобразовании типа в целый из `float` или `double`

```
//  Ошибка на этапе компиляции (CS0220):
//  (2^31 - 1) = 2147483647 - максимальное значение int
int i = 2147483647 + 10;
```

```
//  Нет ошибки. В консоль выводится результат: -2147483639
int ten = 10;
int i = 2147483647 + ten;

Console.WriteLine(i);
```

```
//  Нет ошибки. В консоль выводится результат: -2147483648
double d = 2147483650;
int i = (int)d;

Console.WriteLine(i);
```

# КОНТЕКСТ КОНТРОЛЯ ЗА ПЕРЕПОЛНЕНИЕМ

- checked
- unchecked

# КОНТЕКСТ КОНТРОЛЯ ЗА ПЕРЕПОЛНЕНИЕМ

- В checked контексте арифметическое переполнение приводит к генерации исключения `OverflowException`
- В unchecked контексте переполнение игнорируется и из результата операции обрезаются биты высокого порядка, не умещающиеся в тип результата

# КОНТЕКСТ КОНТРОЛЯ ЗА ПЕРЕПОЛНЕНИЕМ

- По умолчанию контекст unchecked
- Сменить контекст можно глобально (опция компиляции `-checked`), либо локально (ключевые слова `checked` и `unchecked`)

# СИНТАКСИС CHECKED

```
checked(<выражение>)

checked
{
}
```

```
int ten = 10;
Console.WriteLine(checked(2147483647 + ten));

checked
{
    int i = 2147483647 + ten;
    Console.WriteLine(i);
}
```

```
int z = 0;
try
{
    z = checked(maxIntValue + 10);
}
catch (System.OverflowException e)
{
    Console.WriteLine("CHECKED and CAUGHT:  " + e.ToString());
}
Console.WriteLine(z);
```

# СИНТАКСИС UNCHECKED

```
unchecked(<выражение>)

unchecked
{
}
```

```
int i;
unchecked
{
    i = 2147483647 + 10;
}
i = unchecked(2147483647 + 10);
```

```
checked
{
    //  Код, в котором возможно переполнение
    unchecked
    {
        //  Код, где, как вы уверены, не будет переполнения.
        //  Эффективнее его выполнять в unchecked контексте.
    }
    //  Снова код в checked контексте
}
```

```
void Method1()
{
    checked
    {
        Method2();
    }
}

void Method2()
{
    unchecked
    {

    }
}
```

# МАССИВЫ

# МАССИВЫ

- Одномерные
- Многомерные
- Массивы массивов

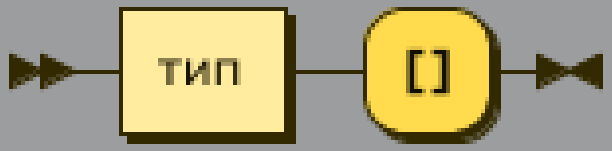
# МАССИВЫ

- Содержат непрерывную последовательность однотипных элементов
- Неявно наследуют от Array <— Object
- Ссылочные типы
- Не могут изменять размер после создания

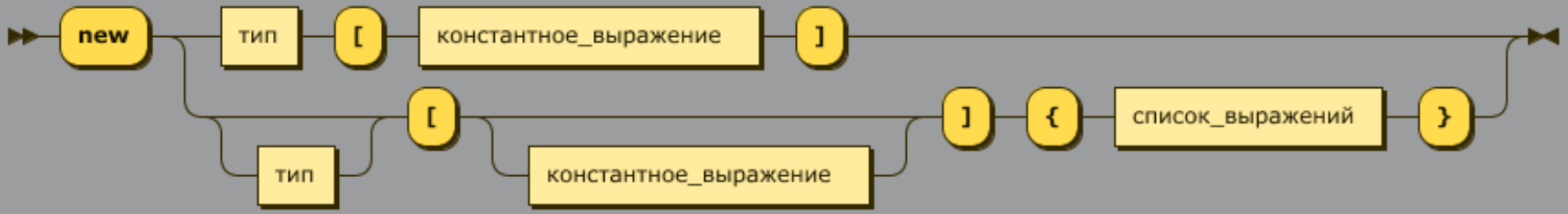
# ОДНОМЕРНЫЕ МАССИВЫ

```
указание_типа_массива
    ::= тип '[]'

инициализация_массива
    ::= 'new' (тип '[' константное_выражение ']'
        | тип? '[' константное_выражение? ']' '{' список_выражений '}')

список_выражений
    ::= выражение | выражение ',' список_выражений
```

указание_типа_массива:



# инициализация_массива:



# ИНИЦИАЛИЗАЦИЯ

```
//  объявление массива
int[] array;

//  инициализация массива из 7 элементов
array = new int[7];

//  инициализация массива из 3 элементов
array = new [] { 3, 7, 15 };
```

# ИНИЦИАЛИЗАЦИЯ

```
//  инициализация массива из 3 элементов
array = (int[])Array.CreateInstance(typeof(int), 3);

//  инициализация пустого массива
array = Array.Empty<int>();
```

# РАБОТА С МАССИВОМ

```
int[] array = new int[4];

for (int i = 0; i < array.Length; i++)
{
    array[i] = i * 3;
}

for (int i = 0; i < array.Length; i++)
{
    Console.WriteLine("array[{0}] = {1}", i + 1, array[i]);
}
```

# РАБОТА С МАССИВОМ

```
int[] array2 = new int[7];

//  Скопировать элементы массива в другой массив начиная с 4 номера во 2 м
array.CopyTo(array2, 3);

//  Скопировать массив
int[] array3 = (int[])array.Clone();

//  Сменить последовательность элементов на обратную
Array.Reverse(array);
```

# РАБОТА С МАССИВОМ

```
//  Заполнить нулями
Array.Fill(array, 0);

//  Установить значение по умолчанию элементам со 2-го по 4-й
Array.Clear(array, 1, 3);

//  Найти порядковый номер элемента со значением 4
Array.IndexOf(array, 4);

//  Найти последний порядковый номер элемента со значением 4
Array.LastIndexOf(array, 4);
```

# РАБОТА С МАССИВОМ

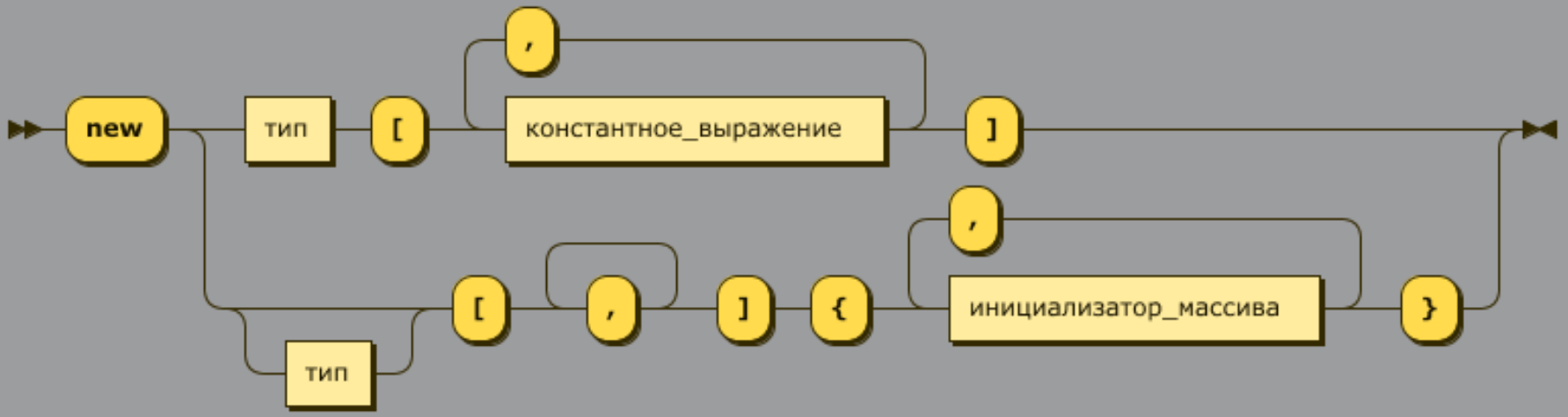
```
//  Сменить размер массива: выделить новую память и скопировать
Array.Resize(ref array, 17);

//  Сортирует элементы, реализующие IComparable
Array.Sort(array);

//  Бинарный поиск в отсортированном заранее массиве
int index = Array.BinarySearch(array, 5);
```

# МНОГОМЕРНЫЕ МАССИВЫ

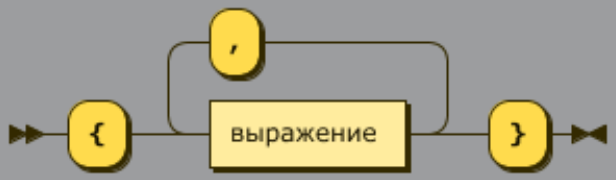
```
указание_типа_многомерного_массива
    ::= тип '[' ','+ ']'

инициализация_многомерного_массива
    ::= 'new' (тип '[' список_константных_выражений ']'
        | тип? '[' ','+ ']' '{' список_инициализаторов_массивов '}')

список_константных_выражений
    ::= константное_выражение | константное_выражение ',' список_константн

список_инициализаторов_массивов
    ::= инициализатор_массива | инициализатор_массива ',' список_инициализ

инициализатор_массива
    ::= '{' список_выражений '}'
```

указание_типа_многомерного_массива:

# инициализация_многомерного_массива:



инициализатор_массива:



# ИНИЦИАЛИЗАЦИЯ

```
//  объявление матрицы
float[,] matrix;

//  инициализация матрицы 3x4
matrix = new float[3, 4];

//  инициализация матрицы 2x3
matrix = new[,] { { 1.0f, 2.0f, 3.0f }, { 4.0f, 5.0f, 6.0f } };

//  инициализация 7-мерного массива
float[,,,,,,] multidimArray = new float[3, 4, 5, 7, 9, 10, 12];
```

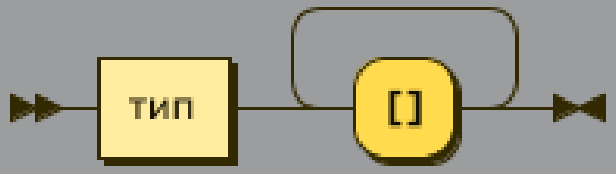
# РАБОТА С МАССИВОМ

```
//  прямой доступ к элементам
matrix[1, 2] = 17.4f;

//  общее число элементов в массиве
int totalLength = matrix.Length;

//  число элементов в первом измерении массива
int rowCount = matrix.GetLength(0);

//  число элементов во втором измерении массива
int columnCount = matrix.GetLength(1);
```

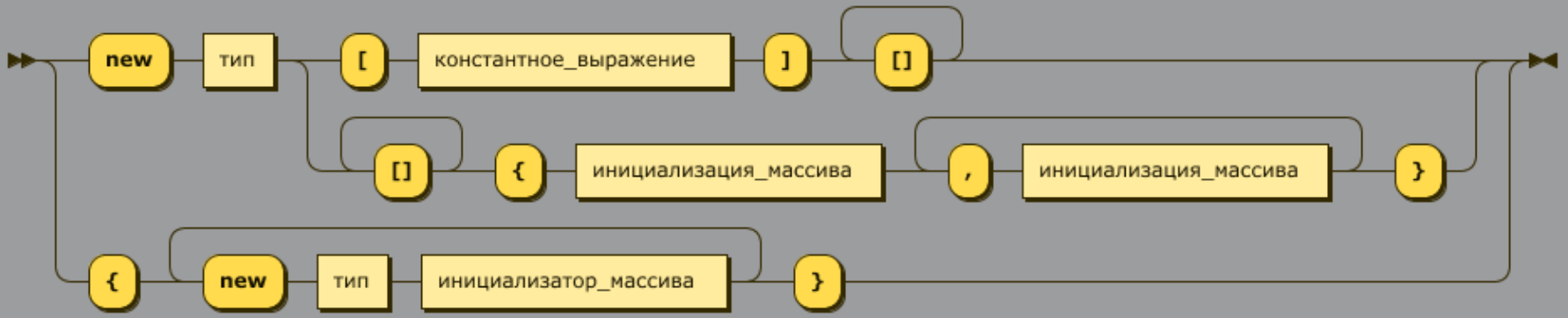
# МАССИВЫ МАССИВОВ

```
указание_типа_массива_массивов
    ::= тип '[]'+

инициализация_массива_массивов
    ::= 'new' (тип '[' константное_выражение ']' '[]'+
        | тип '[]'+ '{' инициализация_массива (',' инициализация_массива)+

инициализация_массива_массивов
    ::= (
            'new' (тип '[' константное_выражение ']'
            | тип? '[' константное_выражение? ']' '{' список_выражений '}'
        )
        | '{' 'new' тип инициализатор_массива '}'

инициализатор_массива
    ::= '{' список_выражений '}'
```

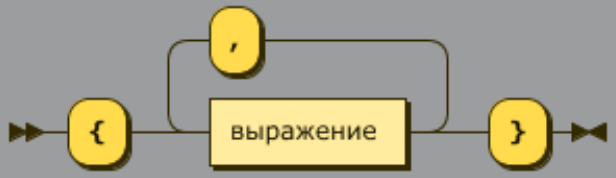
указание_типа_массива_массивов:



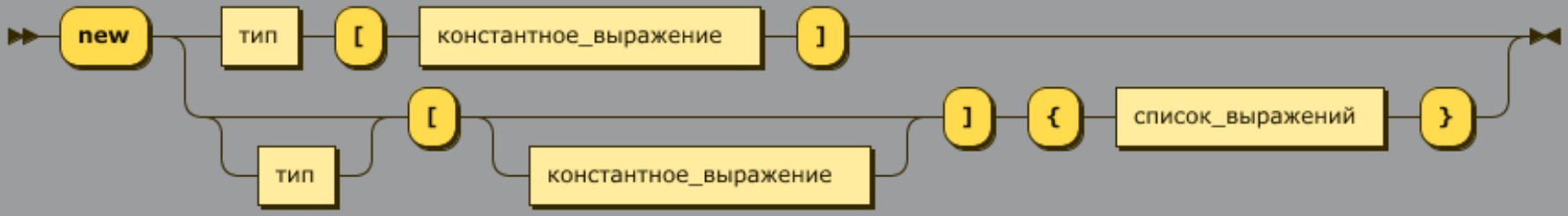
# инициализация_массива_массивов:



инициализатор_массива:



# инициализация_массива:



# ИНИЦИАЛИЗАЦИЯ

```
double[][] jaggedMatrix;

jaggedMatrix = new double[3][];
jaggedMatrix[0] = new double[2];
jaggedMatrix[1] = new double[5];
jaggedMatrix[2] = new double[7];

int[][][][] jaggedMultidimArray = new int[4][][][];
```

# ИНИЦИАЛИЗАЦИЯ

```
double[][] jaggedMatrix;

jaggedMatrix = new double[][]
{
    new [] { 2.3, 4.6 },
    new [] { 1.7, 8.4, 5.6 }
};
```

# ИНИЦИАЛИЗАЦИЯ

```
//  Только одновременное объявление и инициализация
double[][] jaggedMatrix2 =
{
    new double[] { 2.3, 4.6 },
    new double[] { 1.7, 8.4, 5.6 }
};
```

# РАБОТА С МАССИВОМ

```
//  прямой доступ
jaggedMatrix[1][2] = 6.4;

//  число элементов в первом измерении массива
int columnCount = jaggedMatrix.Length;

//  число элементов во втором измерении массива
int rowCount = jaggedMatrix[0].Length;
```

# СПИСОК

System.Collections.Generic.List<T>

System.Collections.Generic.List<T>

- Коллекция элементов
- Размер не лимитирован
- Прямой доступ к элементам

```
// создание экземпляра
var list = new List<int>();

//  добавление элементов
list.Add(3);
list.Add(7);

//  прямой доступ
list[0] = 19;
list[1] = list[0] + 18;

//  число элементов
for (int i = 0; i < list.Count; i++)
    Console.WriteLine(list[i]);
```

```
// удаление элементов по порядковому номеру
list.RemoveAt(1);

// удаление элементов по значению
bool wasRemoved = list.Remove(18);

// удаление ряда элементов - 12 штук начиная с 4-го
list.RemoveRange(3, 12);

// очистка
list.Clear();

// проверка наличия элементов
bool hasSeven = list.Contains(7);
```

## Особенности работы List<T>

- "Под капотом" хранит элементы в массиве
- Исходный размер массива по умолчанию — 4 элемента
- При превышении размера массива - пересоздаёт его с большим размером
- Размер растёт в 2 раза

```
// создание экземпляра с указанием capacity (размера массива)
var list = new List<int>(1200);
```

# СЛОВАРЬ

System.Collections.Generic.Dictionary<TKey, TValue>

System.Collections.Generic.Dictionary<TKey, TValue>

- Коллекция пар "ключ-значение"
- Ассоциативный массив - упорядочен по хэшам ключей
- Хеш вычисляется в методе GetHashCode()
- Быстрый поиск
- Возможны коллизии

```
// создание экземпляра
var dict = new Dictionary<string, string>();

// установка значения по ключу
dict["apple"] = "яблоко";

// получение значения по ключу
string value = dict["apple"];

// проверка наличия значения
bool contains = dict.ContainsKey("apple");
```

# ВОПРОСЫ