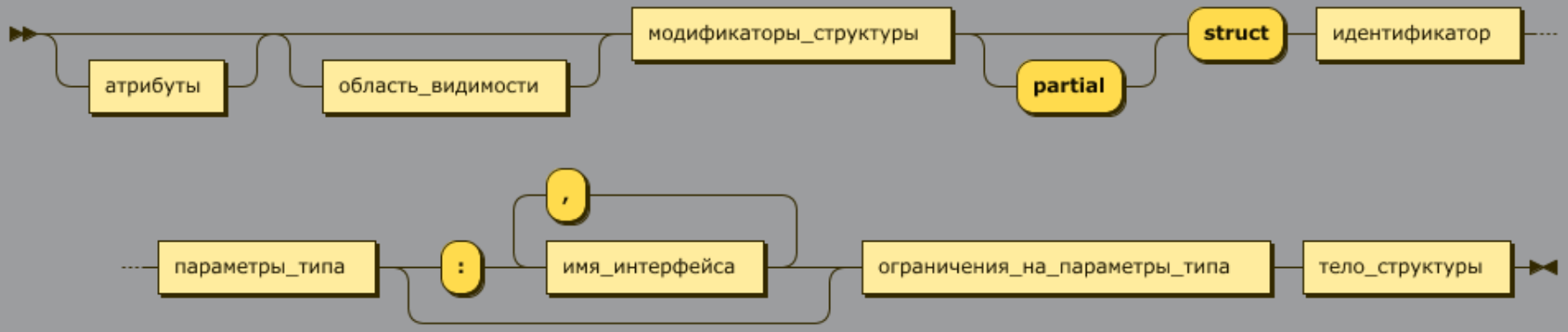
СТРУКТУРЫ

C# - ЛЕКЦИЯ 6

СТРУКТУРЫ

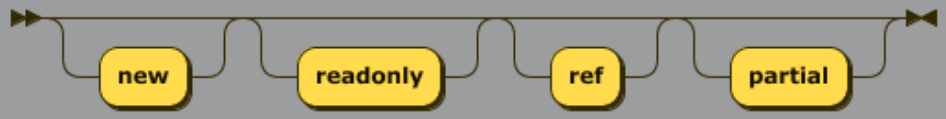
```
объявление_структуры
    ::= атрибуты? область_видимости? модификаторы_структуры 'partial'?
        'struct' идентификатор
        параметры_типа (':' список_интерфейсов)? ограничения_на_параметры_
        тело_структуры
модификаторы_структуры ::= 'new' | 'unsafe'
список_интерфейсов ::= имя_интерфейса | имя_интерфейса ',' список_интерфей
тело_структуры ::= '{' объявление_члена_структуры* '}'
объявление_члена_структуры
    ::= объявление_конструктора | объявление_константы |
        объявление_поля | объявление_свойства |
        объявление_метода | объявление_события |
        объявление_оператора | объявление_индексатора |
        объявление_типа
```

объявление_структуры:

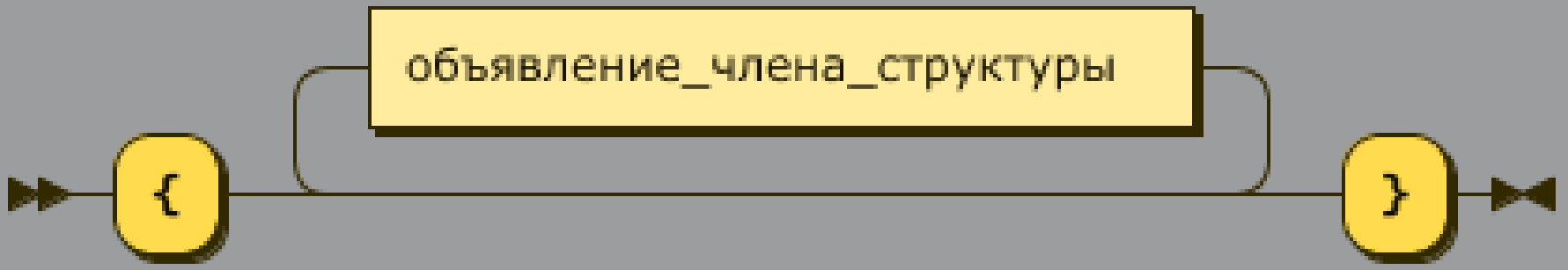


```
модификаторы_структуры
    ::= 'new'? 'readonly'? 'ref'? 'partial'?
```

модификаторы_структуры:



тело_структуры:

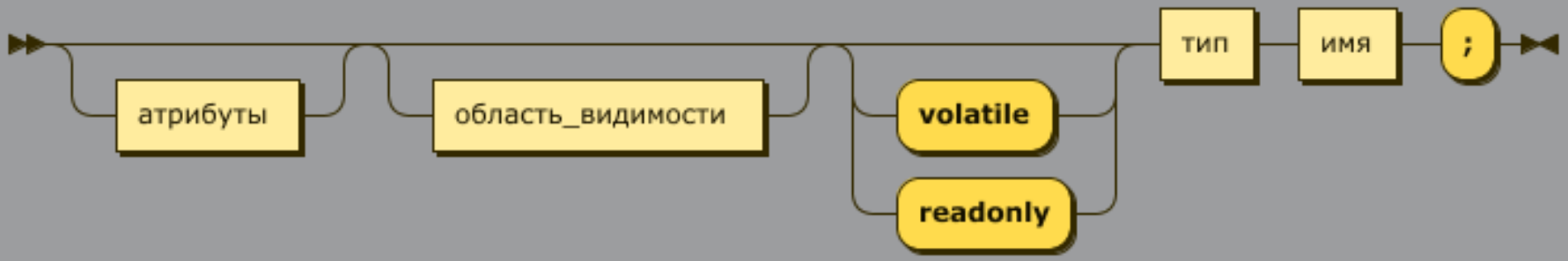


объявление_члена_структуры:

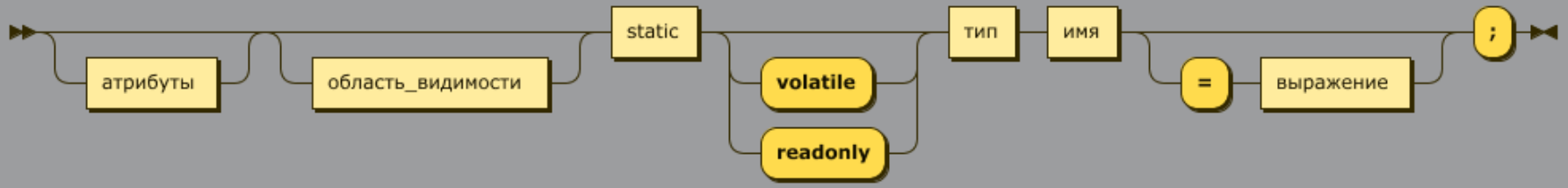
поля

```
объявление_поля_экземпляра_структуры
    ::= атрибуты? область_видимости? ('volatile' | 'readonly')? тип имя';'

объявление_поля_типа_структуры
    ::= атрибуты? область_видимости? static ('volatile' | 'readonly')? тип
        ('=' выражение)? ';'
```

объявление_поля_экземпляра_структуры:



объявление_поля_типа_структуры:



СИНТЕТИЧЕСКИЙ ПРИМЕР

```
struct S
{
    int field1;

    readonly int field2;
}
```

СИНТЕТИЧЕСКИЙ ПРИМЕР

```
struct S
{
    static int field4;

    static readonly int field5 = 7;
}
```

СИНТЕТИЧЕСКИЙ ПРИМЕР

```
struct S
{
    static int a;

    static int b = a + 1;
}
```

СИНТЕТИЧЕСКИЙ ПРИМЕР

```
struct S
{
    //  Ошибки нет
    static int a = b + 1;

    static int b = a + 1;
}
```

СИНТЕТИЧЕСКИЙ ПРИМЕР

```
struct Node
{
    double value;

    //  Ошибка компиляции
    Node next;
}
```

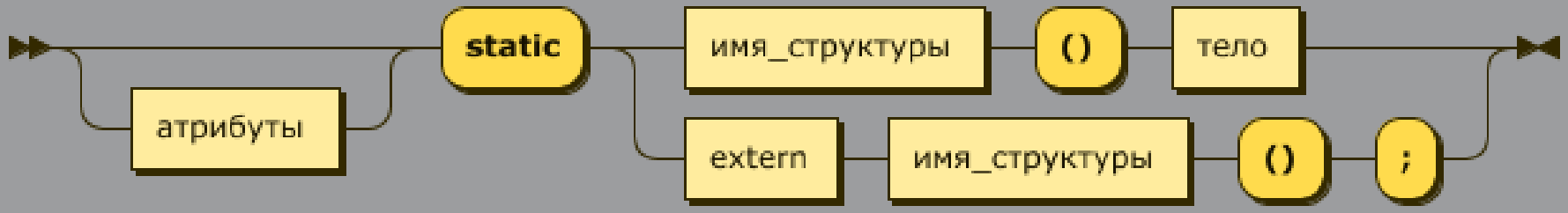
КОНСТРУКТОРЫ СТРУКТУР

- Конструктор типа (статический)
- Конструкторы экземпляров

СТАТИЧЕСКИЙ КОНСТРУКТОР

```
объявление_конструктора_типа_структуры
    ::= атрибуты? 'static' (имя_структуры '()' тело | extern имя_структуры
```

объявление_конструктора_типа_структуры:



```
struct S
{
    static S()
    {
    }
}
```

Статический конструктор

- Вызывается автоматически
- Вызов гарантирован строго до первого обращения к типу
- Вызывается ровно 1 раз
- Если генерирует ошибку, то тип недоступен

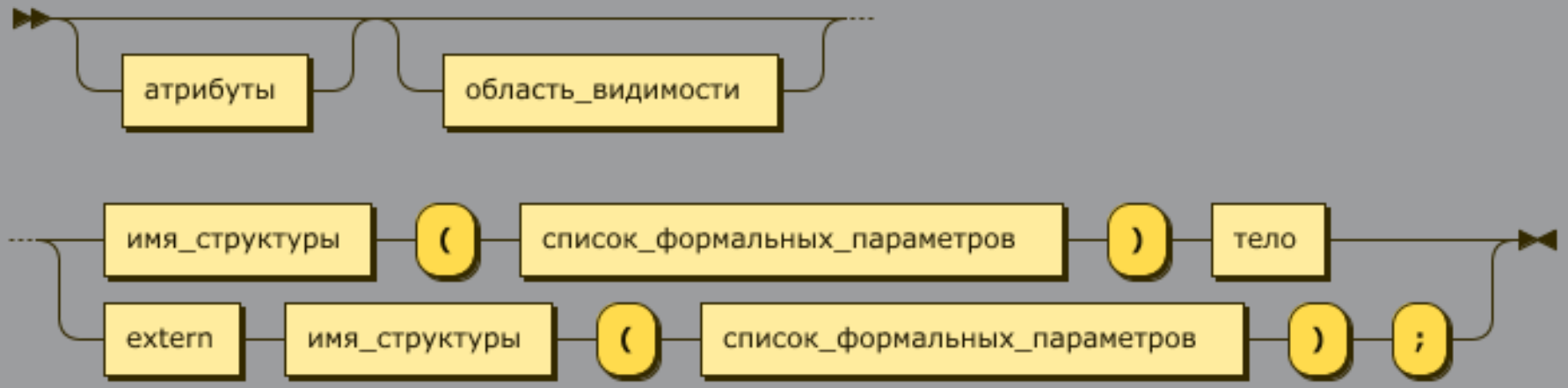
Применение статического конструктора

- При работе с неуправляемым кодом - для загрузки функций из внешней библиотеки
- Проверки во время исполнения: конфигурация, лицензия и т.д.
- Настройка логгирования

КОНСТРУКТОРЫ ЭКЗЕМПЛЯРОВ СТРУКТУР

```
объявление_конструктора_экземпляра_структуры
    ::= атрибуты? область_видимости? (имя_структуры '(' список_формальных_
        | extern имя_структуры '(' список_формальных_параметров? ')' ';')
```

объявление_конструктора_экземпляра_структуры:



КОНСТРУКТОР ЭКЗЕМПЛЯРОВ СТРУКТУР

- Не может быть без параметров
- Обязан инициализировать все поля структуры

```
struct Point
{
    public int X;

    public int Y;

    public Point(int x, int y)
    {
        X = x;
        Y = y;
    }
}
```

```
struct Point
{
    public int X;

    public int Y;

    public int Z;

    // ошибка: поле Z не инициализировано в конструкторе
    public Point(int x, int y)
    {
        X = x;
        Y = y;
    }
}
```

КОНСТАНТЫ

Константы

- Значение известно на этапе компиляции
- Всегда статические, но модификатор static не указывается

```
объявление_константы
    ::= атрибуты? область_видимости? 'const' тип имя '=' константное_выраж
```

объявление_константы:

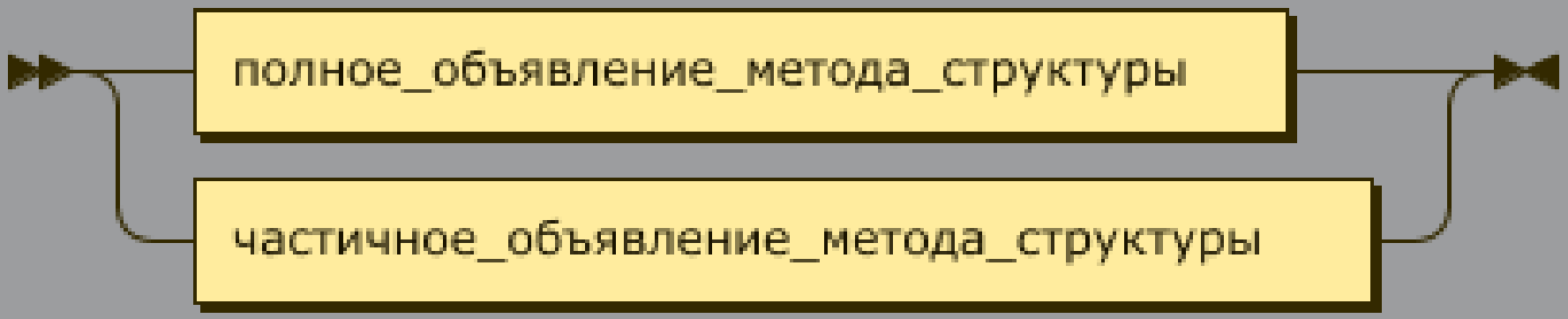
```
struct HumanAge
{
    public const int MaxAge = 117;

    //   ...
}
```

```
struct A
{
    public const double TwoPi = Math.PI * 2;
}
```

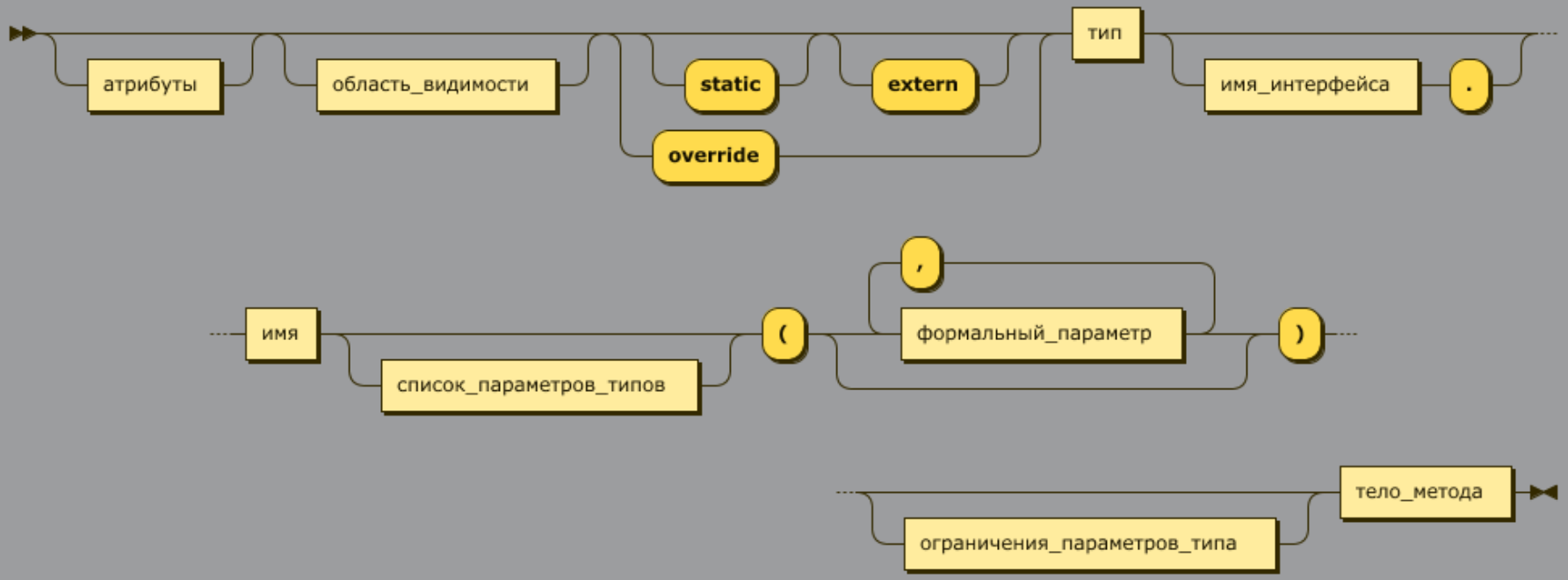
МЕТОДЫ СТРУКТУР

```
объявление_метода_структуры
    ::= полное_объявление_метода_структуры | частичное_объявление_метода_с

полное_объявление_метода_структуры
    ::= атрибуты? область_видимости? ('static'? 'extern'? | 'override')
        тип (имя_интерфейса '.')? имя список_параметров_типов?
        '(' список_формальных_параметров? ')' ограничения_параметров_типов

частичное_объявление_метода_структуры
    ::= атрибуты? 'partial' 'void' имя '(' ')' (';' | тело)

список_формальных_параметров
    ::= формальный_параметр | формальный_параметр ',' список_формальных_па

формальный_параметр
    ::= атрибуты? (('ref' | 'out' | 'in')? тип имя
```

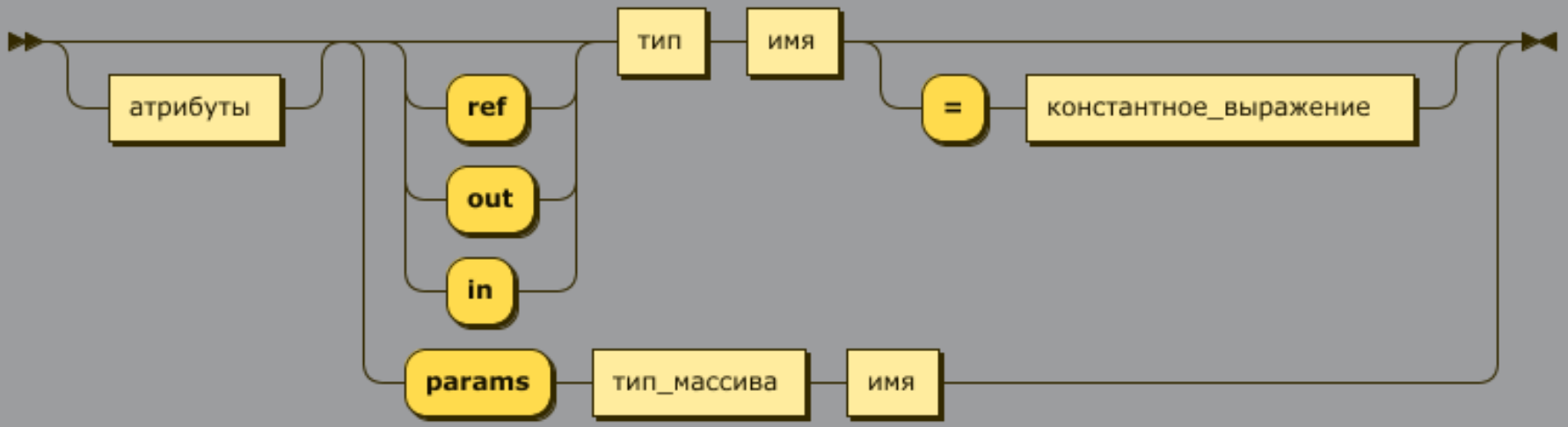
объявление_метода_структуры:



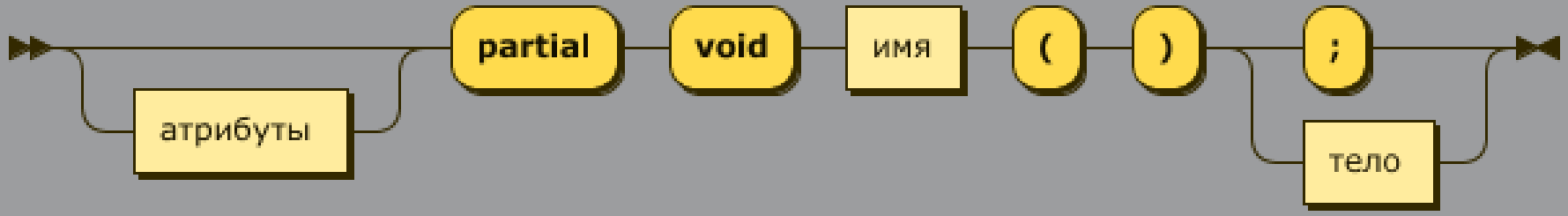
полное_объявление_метода_структуры:



формальный_параметр:



частичное_объявление_метода_структуры:



```
struct Complex
{
    public double real;

    public double imaginary;

    public void Add(Complex value)
    {
        real += value.real;
        imaginary += value.imaginary;
    }
}
```

```
struct Complex
{
    public double real;

    public double imaginary;

    public double GetModulus()
    {
        return Math.Sqrt(real * real + imaginary * imaginary);
    }
}
```

```
struct Complex
{
    public double real;

    public double imaginary;

    public static Complex Sum(Complex a, Complex b)
    {
        return new Complex
        {
            real = a.real + b.real,
            imaginary = a.imaginary + b.imaginary
        };
    }
}
```

Модификатор extern

- Указывает, что метод имеет внешнюю реализацию
- Обычно локация реализации указывается атрибутом DllImport

```
struct S
{
    [DllImport("User32.dll", CharSet=CharSet.Unicode)]
    public static extern int MessageBox(IntPtr h, string m, string c, int t
}
```

Модификатор override

- Указывает, что метод переопределяет базовую реализацию
- Применим только к виртуальным методам - помеченным модификатором virtual

```
struct Complex
{
    public double real;

    public double imaginary;

    public override string ToString()
    {
        return $"{real} + {imaginary}i";
    }
}
```

Модификатор partial

- Указывает, что метод определён в другой части определения типа
- Может только иметь сигнатуру вида partial void имя()

```
partial struct B
{
    partial void DoWork();
}

partial struct B
{
    partial void DoWork()
    {
        Console.WriteLine("Working...");
    }

}
```

Имя интерфейса перед именем метода

- Это явная реализация интерфейса
- Рассмотрим в разделе "интерфесы" одной из следующих лекций

Список параметров типов и ограничения параметров типов

- Это шаблонные методы
- Рассмотрим в разделе "универсальные шаблоны" одной из следующих лекций

ref, out и in параметры

- Параметры передаются по ссылке
- ref параметр может быть изменён в теле метода
- out параметр обязан быть изменён в теле метода
- in параметр не может быть изменён в теле метода

```
private static void ChangeByReference(ref Product itemRef)
{
    itemRef = new Product("Stapler", 99999);

    itemRef.ItemID = 12345;
}
```

```
double d;

if (double.TryParse(Console.ReadLine(), out d));
    ...
```

```
private static double CalculateDistance3(in Point3D point1, in Point3D poi
{
    double xDifference = point1.X - point2.X;
    double yDifference = point1.Y - point2.Y;
    double zDifference = point1.Z - point2.Z;

    return Math.Sqrt(xDifference * xDifference +
        yDifference * yDifference + zDifference * zDifference);
}
```

модификатор params у параметра

- Указывает, что он принимает переменное количество значений: 0...*
- типом параметра должен быть массив
- при вызове можно передать значения через запятую
- при вызове можно передать массив

```
static int Sum(params int[] numbers)
{
    int result = 0;

    foreach (var item in numbers)
    {
        result += item;
    }

    return result;
}
```

```
static void Main(string[] args)
{
    int r1 = Sum(1, 2, 3, 4, 5);
    int r2 = Sum(new[] { 1, 2, 3, 4, 5 });
}
```

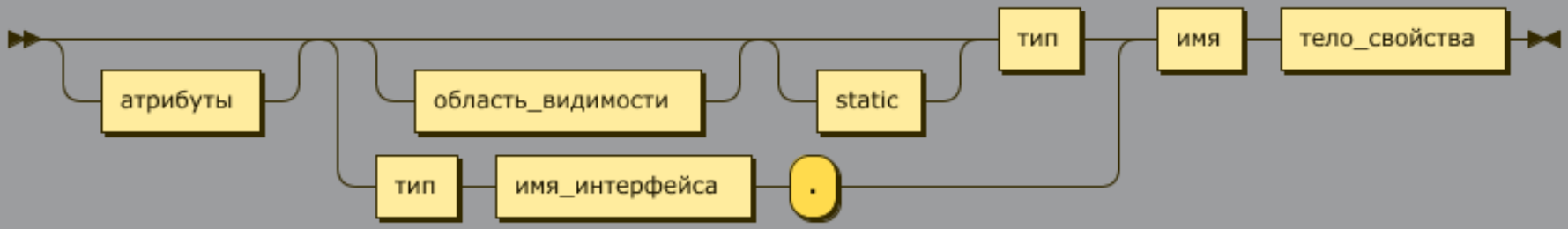
СВОЙСТВА

Свойства

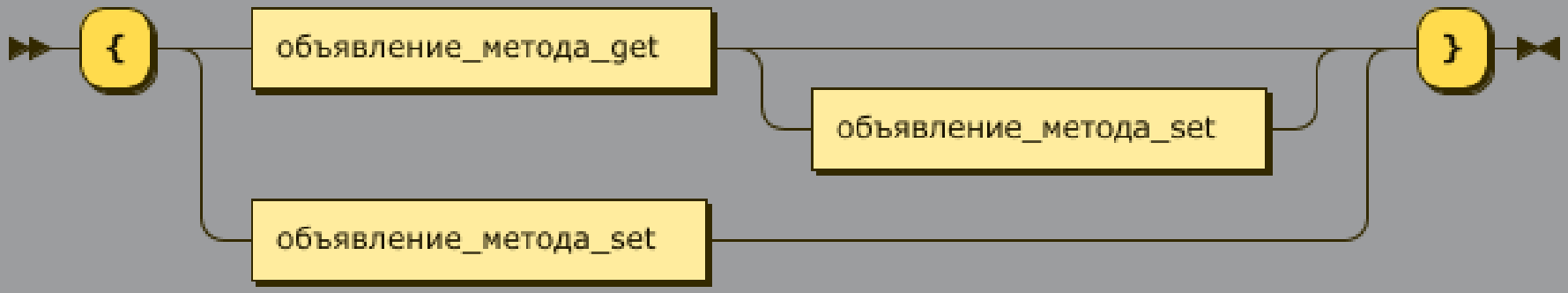
- Синтаксически аналогичны полям для пользователя
- Позволяют определить произвольную реализацию для чтения и записи
- Позволяют ограничить или запретить чтение или запись

```
объявление_свойства_структуры
    ::= атрибуты? (область_видимости? static? тип | тип имя_интерфейса '.'

тело_свойства
    ::= '{' (объявление_метода_get объявление_метода_set
        | объявление_метода_get | объявление_метода_set) '}'

объявление_метода_get
    ::= атрибуты? область_видимости? 'get' тело_метода_get

объявление_метода_set
    ::= атрибуты? область_видимости? 'set' тело_метода_set
```

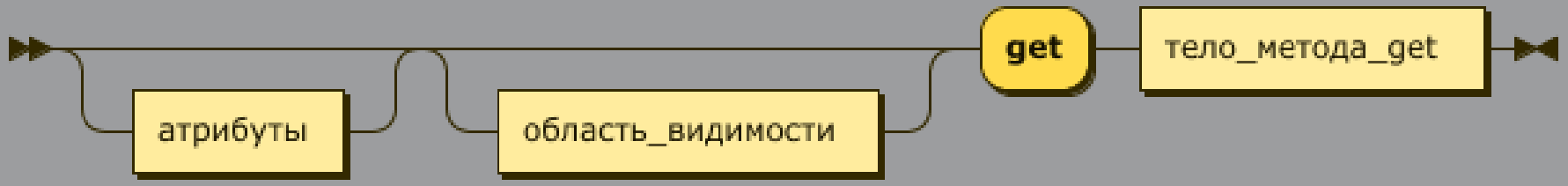
объявление_свойства_структуры:



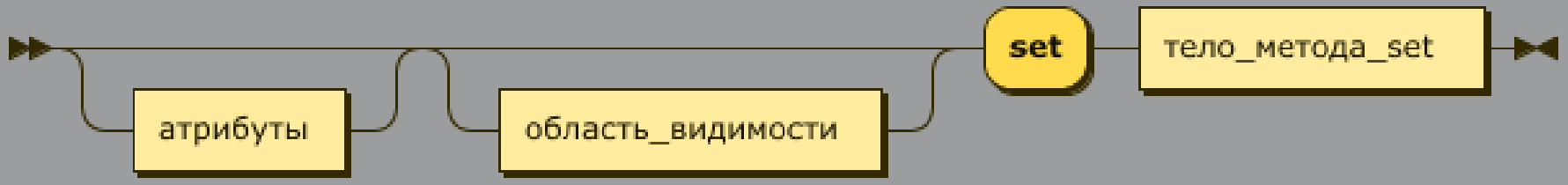
тело_свойства:



объявление_метода_get:



объявление_метода_set:



тело_метода_get

- Соответствует телу метода, не принимающего параметров и возвращающего значение типа данного свойства
- Область видимости может только сужать область видимости свойства

тело_метода_set

- Соответствует телу метода, принимающего параметр с именем value и типом данного свойства и не возвращающего значение
- Область видимости может только сужать область видимости свойства

```
struct Meter
{
    private double value;

    public double Value
    {
        get { return value; }
        set { this.value = value; }
    }
}
```

```
struct Meter
{
    private double value;

    public double Value
    {
        get { return value; }
    }
}
```

```
struct Meter
{
    private double value;

    public double Value
    {
        set { this.value = value; }
    }
}
```

```
struct Meter
{
    private double value;

    public double Value
    {
        get { return value; }
        internal set { this.value = value; }
    }
}
```

Автоматически-реализуемые свойства

- Это синтаксический сахар
- При компиляции преобразуются в свойство, читающее и записывающее значение поля

Автоматически-реализуемое свойство Y

```
struct Point
{
    private int x;

    public int X
    {
        get { return x; }
        set { x = value; }
    }

    public int Y { get; set; }
}
```

Автоматически-реализуемые readonly свойства

- Это синтаксический сахар
- При компиляции преобразуются в свойство, читающее значение readonly поля

Автоматически-реализуемые readonly свойства

```
struct Point
{
    public int X { get; }

    public int Y { get; }

    public Point(int x, int y)
    {
        X = x;
        Y = y;
    }
}
```

ВОПРОСЫ