КЛАССЫ

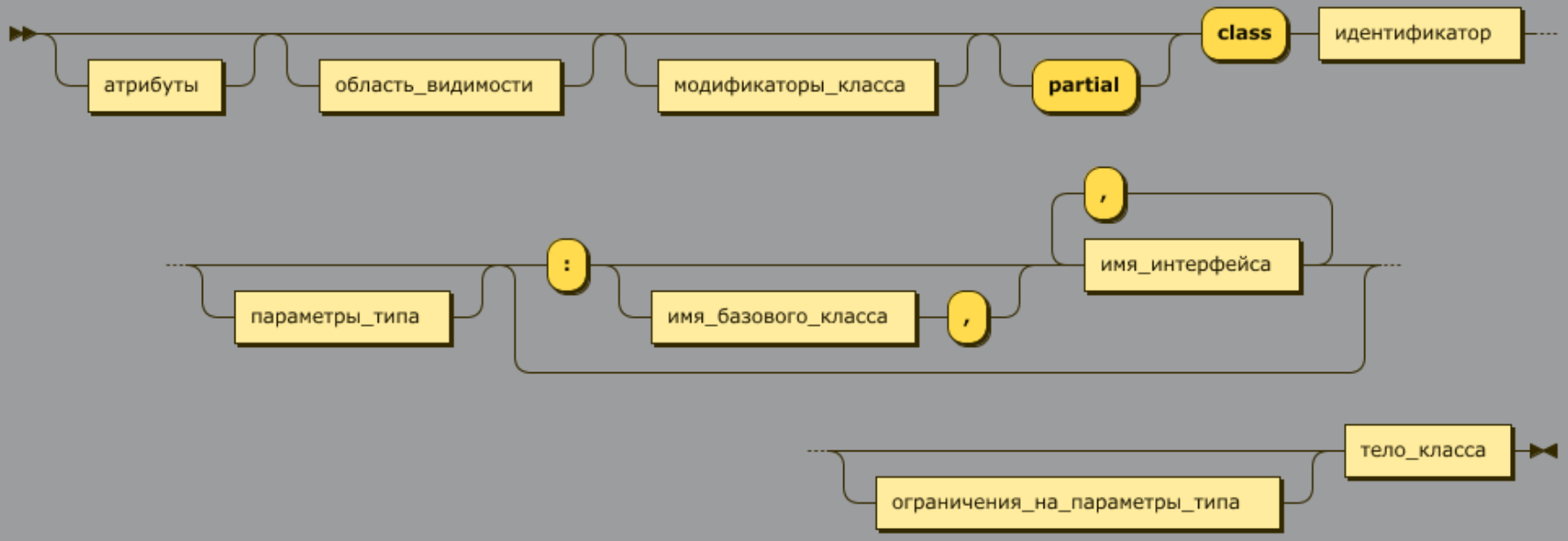
C# - ЛЕКЦИЯ 8

ОБЪЯВЛЕНИЕ КЛАССА

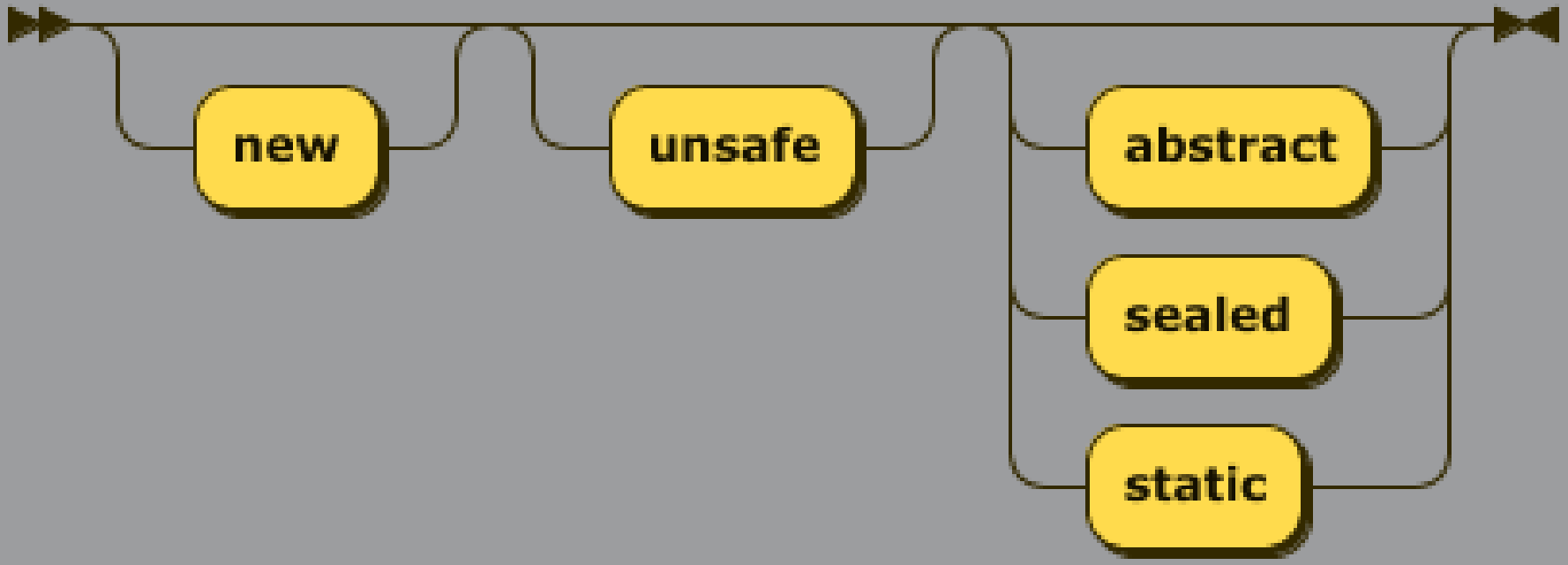
СИНТАКСИС

```
объявление_класса
    ::= атрибуты? область_видимости? модификаторы_класса? 'partial'?
        'class' идентификатор
        параметры_типа? (':' (имя_базового_класса ',' список_интерфейсов |
        ограничения_на_параметры_типа? тело_класса
модификаторы_класса ::= 'new'? 'unsafe'? ('abstract' | 'sealed' | 'static'
список_интерфейсов ::= имя_интерфейса | имя_интерфейса ',' список_интерфей
тело_класса ::= '{' объявление_члена_класса* '}'
объявление_члена_класса
    ::= объявление_конструктора | объявление_деструктора | объявление_конс
        объявление_поля | объявление_свойства |
        объявление_метода | объявление_события |
        объявление_оператора | объявление_индексатора | объявление_типа
```

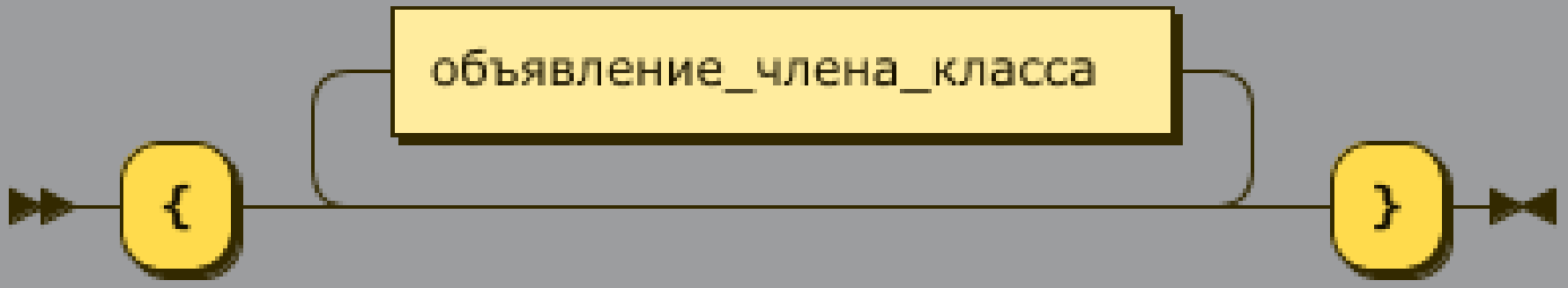
объявление_класса:



модификаторы_класса:



тело_класса:



объявление_члена_класса:

МОДИФИКАТОР NEW

Указывает, что класс подменяет одноимённое определение вложенного типа

МОДИФИКАТОР UNSAFE

Указывает, что тело класса имеет небезопасный контекст:

- Можно использовать указатели
- Необходимо включить опцию компиляции unsafe

МОДИФИКАТОР ABSTRACT

Помечает класс как абстрактный:

- Нельзя создавать его экземпляры
- Можно объявлять в нём абстрактные члены: методы, свойства, индексаторы и события

МОДИФИКАТОР SEALED

Помечает класс как запечатанный:

- Нельзя наследовать от него

МОДИФИКАТОР STATIC

Помечает класс как статический:

- Можно объявлять в нём только статические члены
- Нельзя наследовать от него

ОБЪЯВЛЕНИЕ КЛАССА

ПРИМЕРЫ

```
class Person
{
}
```

```
class Person
{
}

class Employee : Person
{
}

class Customer : Person
{
}
```

```
var p = new Person();
var e = new Employee();
var c = new Customer();

bool b1 = e is Employee;     //  true
bool b2 = e is Person;       //  true
```

ПОЛЯ КЛАССОВ

ОТЛИЧИЕ ОТ ПОЛЕЙ СТРУКТУР

Поля экмемпляров могут иметь инициализаторы

- Выражение в инициализаторе не может обращаться к this (т.е. ни к каким члена экземпляра)

ПОЛЯ КЛАССОВ

ПРИМЕРЫ

```
class ExchangeRate
{
    private Currency sourceCurrency;

    private Currency targetCurrency;

    private decimal rate;

    private DateTime timeStart = DateTime.Now;

    private DateTime timeEnd = DateTime.Now.Add(TimeSpan.FromSeconds(Defau

    public const int DefaultDurationSeconds = 30;

    //  ...
}
```

КОНСТРУКТОРЫ КЛАССОВ

- Статический конструктор
- Конструкторы экземпляров

СТАТИЧЕСКИЙ КОНСТРУКТОР

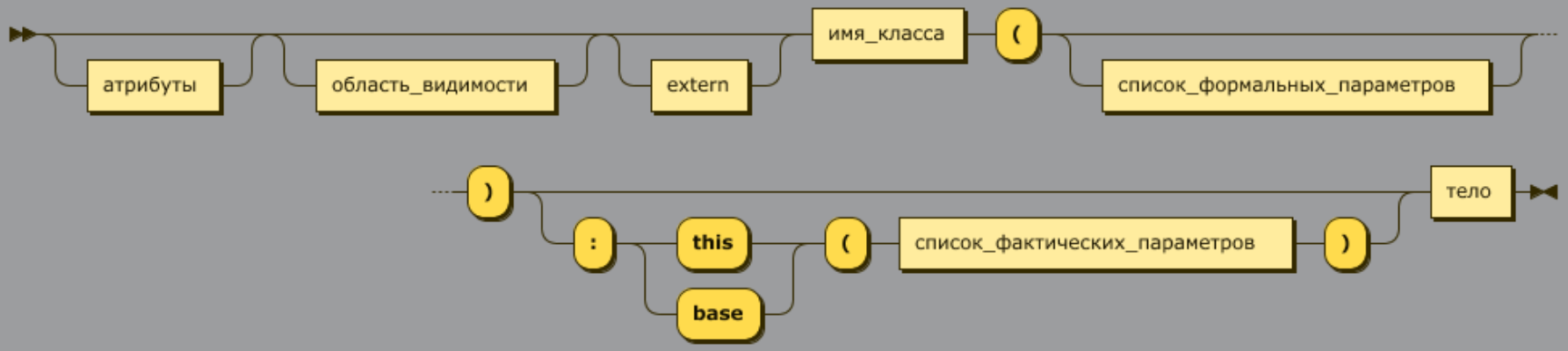
Аналогичен статическому конструктору структуры

КОНСТРУКТОРЫ ЭКЗЕМПЛЯРОВ КЛАССОВ

СИНТАКСИС

```
объявление_конструктора_экземпляра_класса
    ::= атрибуты? область_видимости? extern? имя_класса '(' список_формальн
        (':' ('this' | 'base') '(' список_фактических_параметров ')')? тел
```

объявление_конструктора_экземпляра_класса:



КОНСТРУКТОР ПО УМОЛЧАНИЮ

- Генерируется автоматически
- Публичный, без параметров

КОНСТРУКТОР ПО УМОЛЧАНИЮ

```
class Person
{
    public string Fullname { get; set; }
}

class Program
{
    static void Main()
    {
        var p = new Person();
    }
}
```

КОНСТРУКТОР ПО УМОЛЧАНИЮ

- Не генерируется при объявлении хотя бы одного конструктора

Конструктор по умолчанию отсутствует

```
class Person
{
    public string Fullname { get; }

    public Person(string fullname)
    {
        Fullname = fullname;
    }
}

class Program
{
    static void Main()
    {
        var p = new Person("Peter Peterson");
    }
}
```

Конструктор по умолчанию отсутствует

```
class Person
{
    public Person()
    {
        Console.WriteLine("some text");
    }
}
```

Паттерн "класс-одиночка"

```
class Singleton
{
    private Singleton()
    {
    }

    public static Singleton Instance { get; } = new Singleton();
}
```

Перегрузка конструкторов

```
class Person
{
    public string Fullname { get; }

    public DateTime? BirthDate { get; }

    public Person(string fullname, DateTime birthDate)
        : this(fullname)
    {
        BirthDate = birthDate;
    }

    public Person(string fullname)
    {
        if (string.IsNullOrEmpty(fullname))
            throw new ArgumentException("Fullname shouldn't be empty", nam

        Fullname = fullname;
    }
}
```

ПОРЯДОК ИНИЦИАЛИЗАЦИИ

1. Поля инициализируются значениями по умолчанию
2. Выполняются инициализаторы полей
3. Выполняется инициализатор конструктора (this или base)
4. Выполняется тело конструктора

Порядок инициализации

```
class A
{
    //  1. b и c = dafult(int)
    //  2. b = 7, c = 13
    //  3. this.b = b
    //  4. this.c = c
    private int b = 7;

    private int c = 13;

    public A(int b, int c)
        : this(b)
    {
        this.c = c;
    }

    public A(int b)
    {
        this.b = b;
    }
}
```

НАСЛЕДОВАНИЕ

1. Конструкторы не наследуются
2. Если все конструкторы базового класса имеют параметры, то необходимо их вызывать в наследнике

Наследование конструкторов

```
class Animal
{
    public string Kind { get; }

    public Animal(string kind)
    {
        Kind = kind;
    }
}

class Human : Animal
{
    public Human()
        : base("Homo sapiens")
    {
    }
}
```

МЕТОДЫ КЛАССОВ

СТАТИЧЕСКИЕ МЕТОДЫ
МЕТОДЫ ЭКЗЕМПЛЯРОВ

СТАТИЧЕСКИЕ МЕТОДЫ

Аналогичны статическим методам структур

МЕТОДЫ ЭКЗЕМПЛЯРОВ

Отличия от методов экземпляров структур:

- Модификаторы abstract, virtual, sealed и new
- Поведение this - как readonly поле. В методах экземпляров структур - как ref параметр.

МОДИФИКАТОР VIRTUAL

Делает метод виртуальным

- Можно переопределять в наследнике - override
- Реализация определяется на этапе исполнения исходя из типа экземпляра, для которого производится вызов. Реализация невиртуального метода известна на этапе компиляции.
- Нельзя сделать невиртуальным в наследнике

```
class Worker
{
    public virtual void Work()
    {
        Console.WriteLine("Work, work");
    }
}

class HardWorker : Worker
{
    public override void Work()
    {
        base.Work();
        Console.WriteLine("Work more");
    }
}

class Manager : Worker
{
    public override void Work()
    {
        Console.WriteLine("Manage");
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Worker w = new Worker();
        HardWorker hw = new HardWorker();
        Manager m = new Manager();

        m.Work();   //  Work, work
        hw.Work();  //  Work, work\nWork more
        m.Work();   //  Manage

        Worker w1 = m;

        w1.Work();  //  Manage
    }
}
```

МОДИФИКАТОР ABSTRACT

Делает метод абстрактным

- Он виртуальный
- Он не имеет реализации в текущем классе
- Наследник обязан предоставить реализацию

```
abstract class Shape
{
    public abstract double GetPerimeter();
}

class Triangle : Shape
{
    public Point P1 { get; set; }

    public Point P2 { get; set; }

    public Point P3 { get; set; }

    public override double GetPerimeter()
    {
        return new Line(P1, P2).Length + new Line(P1, P2).Length + new Lin
    }
}
```

```
class A
{
    public virtual void F()
    {
        Console.WriteLine("A.F");
    }
}

abstract class B : A
{
    public abstract override void F();
}

class C : B
{
    public override void F()
    {
        Console.WriteLine("C.F");
    }
}
```

МОДИФИКАТОР SEALED

Делает виртуальный метод запечатанным

- Наследник не может его переопределить - override

```
class A
{
    public virtual void F()
    {
        Console.WriteLine("A.F");
    }
}

class B : A
{
    public sealed override void F()
    {
        Console.WriteLine("B.F");
    }
}

class C : B
{
    public new void F()
    {
        Console.WriteLine("C.F");
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        A a = new A();
        B b = new B();
        C c = new C();

        a.F();  //  A.F
        b.F();  //  B.F
        c.F();  //  C.F

        A a1 = c;
        B b1 = c;

        a1.F(); //  B.F
        b1.F(); //  B.F
    }
}
```

МОДИФИКАТОР NEW

Подменяет одноимённый метод базового класса

- Неважно - виртуальный или нет

```
class Program
{
    static void Main(string[] args)
    {
        Cannon c = new Cannon();
        Gun g = c;

        c.Shoot();  //  Fire a projectile
        g.Shoot();  //  Fire a bullet
    }
}
class Gun
{
    public void Shoot()
    {
        Console.WriteLine("Fire a bullet");
    }
}
class Cannon : Gun
{
    public new void Shoot()
    {
        Console.WriteLine("Fire a projectile");
    }
}
```

СВОЙСТВА КЛАССОВ

СТАТИЧЕСКИЕ СВОЙСТВА

СВОЙСТВА ЭКЗЕМПЛЯРОВ

СТАТИЧЕСКИЕ СВОЙСТВА

Аналогичны статическим свойствам структур

СВОЙСТВА ЭКЗЕМПЛЯРОВ

Отличия от методов экземпляров структур:

- Модификаторы abstract, virtual, sealed и new
- Поведение this - как readonly поле. В методах экземпляров структур - как ref параметр.

ВИРТУАЛЬНЫЕ СВОЙСТВА ЭКЗЕМПЛЯРОВ

- Оба метода (get и set) - виртуальные
- Если в свойстве 1 метод, то в переопределяемом тоже 1. Если 2, то 1 или 2
- Если в свойстве 2 метода, то в переопределяемом 1 или 2

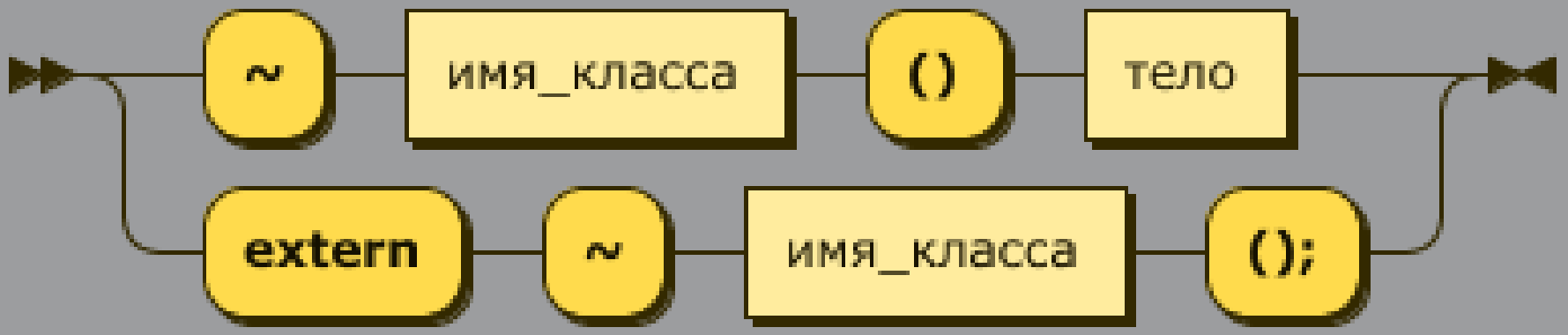
АБСТРАКТНЫЕ СВОЙСТВА ЭКЗЕМПЛЯРОВ

- Оба метода (get и set) - виртуальные
- Если в свойстве 1 метод, то в переопределяемом тоже 1. Если 2, то 1 или 2
- Если в свойстве 2 метода, то в переопределяемом 2

ДЕСТРУКТОРЫ КЛАССОВ

```
объявление_деструктора_класса
    ::= '~' имя_класса '()' тело | 'extern' '~' имя_класса '();'
```

объявление_деструктора_класса:



```
class A
{
     ~A()
     {
     }
}
```

```
class ClassWithUnmanagedResources : IDisposable
{
    private bool disposed = false;

    protected virtual void Dispose(bool disposing)
    {
        if (!disposed)
        {
            if (disposing)
            {   /* TODO: dispose managed state (managed objects).  */ }

            // TODO: free unmanaged resources (unmanaged objects) and over

            disposed = true;
        }
    }

    ~ClassWithUnmanagedResources() { Dispose(false); }

    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }
}
```

ВОПРОСЫ