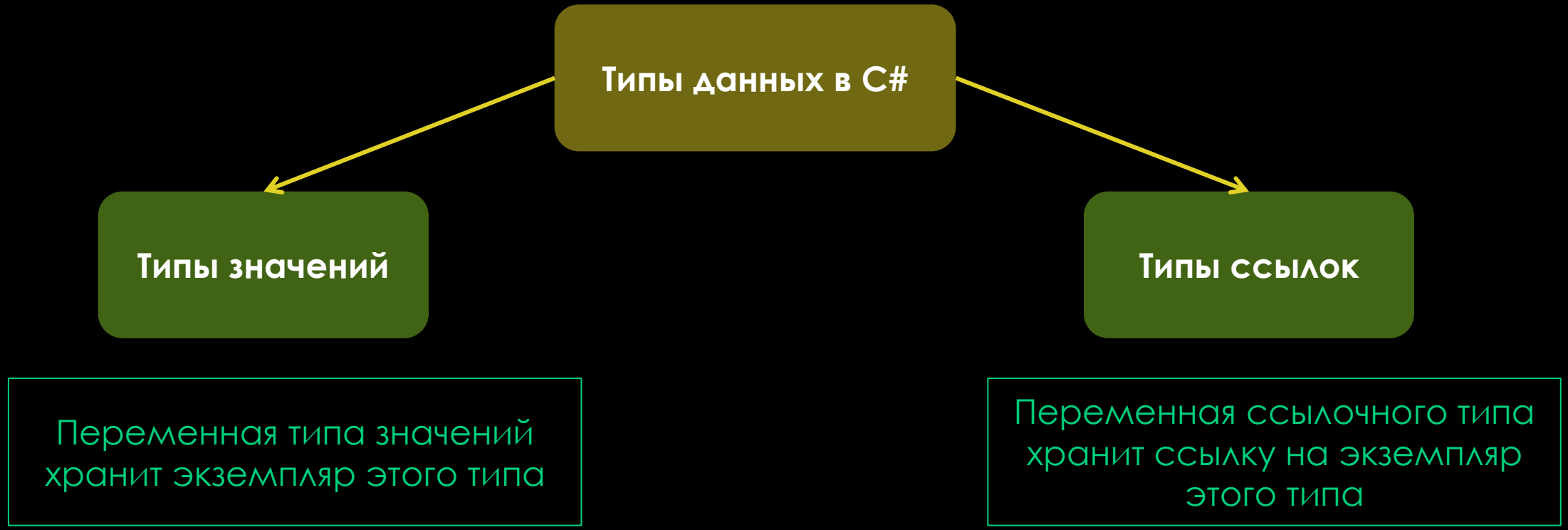
ЛЕКЦИЯ 9

- Модуль 2
- 29.11.2023
- Типы значений

ЦЕЛИ ЛЕКЦИИ

- Получить представления о типе данных структура (struct)
- Получить представление о перечислениях (enum)

ССЫЛОЧНЫЕ ТИПЫ И ТИПЫ ЗНАЧЕНИЙ

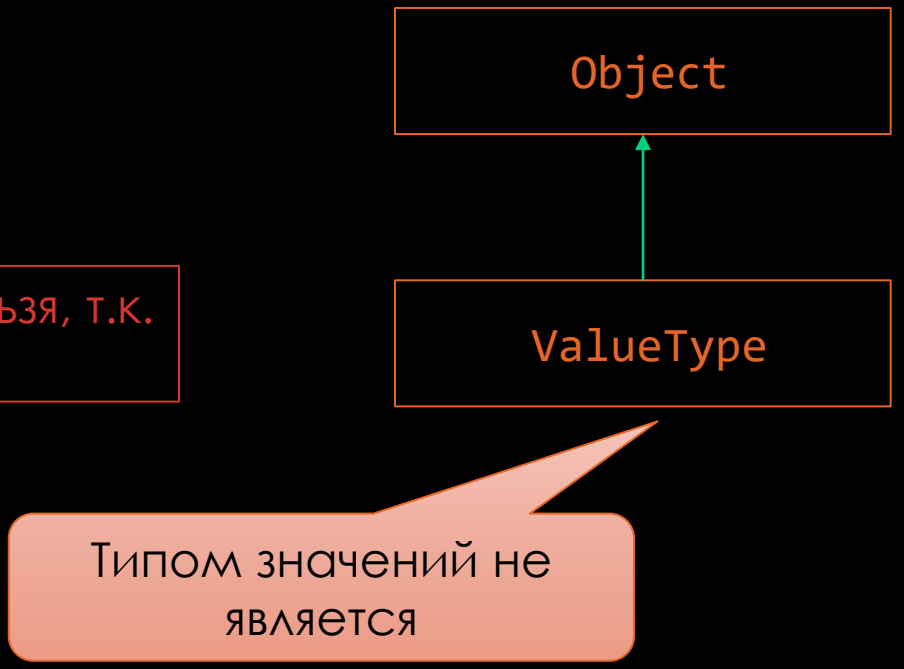


ТИПЫ ЗНАЧЕНИЙ

Пользовательскими типами значений являются:

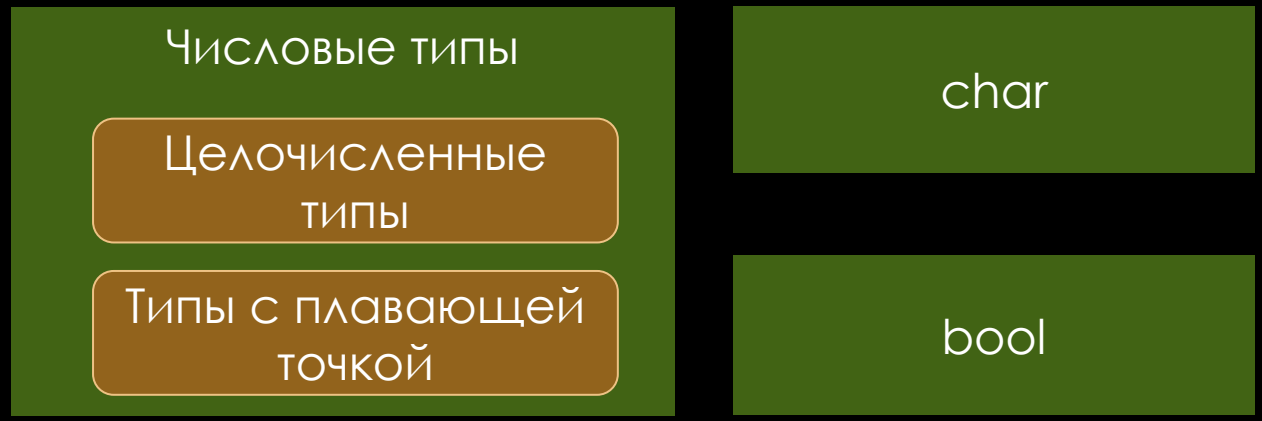
- Структуры (struct) и Записи-Структуры (record struct, C# 10, преобразуются в структуры при компиляции)
- Перечисления (enum)

Напрямую из типов значений у наследоваться нельзя, т.к. они по умолчанию запечатаны (sealed)



ВСТРОЕННЫЕ ТИПЫ ЗНАЧЕНИЙ

- В C# реализованы предопределённые типы структур – **простые типы** [simple types]
- Простые типы идентифицируются ключевыми словами языка C#



Keyword	Aliased type
sbyte	System.SByte
byte	System.Byte
short	System.Int16
ushort	System.UInt16
int	System.Int32
uint	System.UInt32
long	System.Int64
ulong	System.UInt64
char	System.Char
float	System.Single
double	System.Double
bool	System.Boolean
decimal	System.Decimal

КОНСТРУКТОРЫ УМОЛЧАНИЯ

- **Конструктор умолчания** (*default constructor*) без параметров неявно объявлен для всех типов значений
- Возвращает инициализированный нулём (умалчиваемое значение) объект типа значения

Для простых типов

- For `sbyte`, `byte`, `short`, `ushort`, `int`, `uint`, `long`, and `ulong`, the default value is `0`.
- For `char`, the default value is `'\x0000'`.
- For `float`, the default value is `0.0f`.
- For `double`, the default value is `0.0d`.
- For `decimal`, the default value is `0m` (that is, value zero with scale 0).
- For `bool`, the default value is `false`.
- For an *enum_type* `E`, the default value is `0`, converted to the type `E`.

Для структур все поля с типами значений устанавливаются в умалчиваемые значение, а ссылочные в `null`

Для nullable-типа значением по умолчанию будет объект с свойством `HasValue` равным `false`

СТРУКТУРЫ

struct

СТРУКТУРЫ

Структуры – представляют структуры данных, состоящие из членов-данных и функциональных членов, но не требуют размещения в куче

Переменная структурного типа напрямую содержит данные структуры, а не ссылку на них

Синтаксис объявления структур:

```
[Модификаторы] struct <Идентификатор> { [Объявление членов] }
```

Рекомендуется использовать типы структур в качестве легковесных контейнеров для данных, не обладающих сложным поведением

ключевое слово

```
public struct Point2D
{
    public double X;
    public double Y;
}
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ СТРУКТУРЫ

```
public struct Point2D
{
    public double X;
    public double Y;
}
```

Структура **Point2D** – это тип-значений, представляющий две координаты точки на плоскости

```
Point2D first, second, third;
first.X = 10; first.Y = 10;
second.X = 20; second.Y = 20;
third.X = first.X + second.X;
third.Y = first.Y + second.Y;
Console.WriteLine($"First: x = {first.X}, y = {first.Y}");
Console.WriteLine($"Second: x = {second.X}, y = {second.Y}");
Console.WriteLine($"Third: x = {third.X}, y = {third.Y}");
```

Локальные переменные с типом структуры **Point2D**

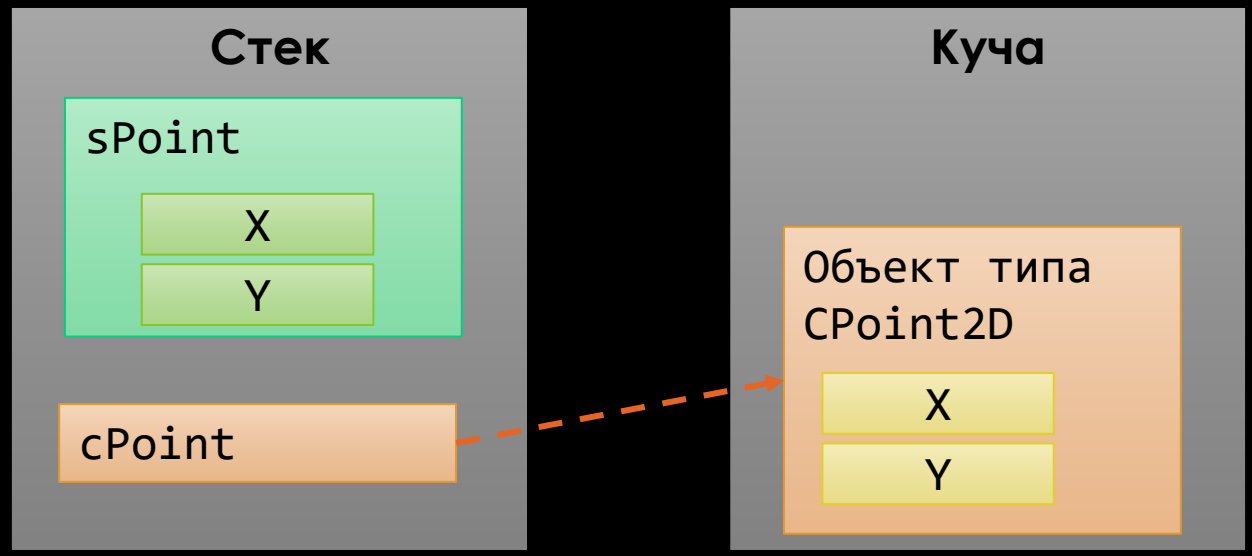
Инициализация полей структур

Вывод:

```
First: x = 10, y = 10
Second: x = 20, y = 20
Third: x = 30, y = 30
```

ХРАНЕНИЕ СТРУКТУРЫ И ОБЪЕКТА КЛАССА В ПАМЯТИ

```
CPoint2D cPoint = new();
SPoint2D sPoint = new();

public class CPoint2D
{
    public int X;
    public int Y;
}

public struct SPoint2D
{
    public int X;
    public int Y;
}
```



ПРИСВАИВАНИЕ ДЛЯ СТРУКТУРЫ И ДЛЯ КЛАССА

```
public class CPoint2D
{
    public int X;
    public int Y;
}
public struct Point2D
{
    public int X;
    public int Y;
}
```

```
CPoint2D cs1 = new CPoint2D(), cs2 = null;
Point2D ss1 = new Point2D(), ss2 = new Point2D();
cs1.X = ss1.X = 5;
cs1.Y = ss1.Y = 10;

// Присваивание для класса - 2 ссылки на 1 объект.
cs2 = cs1;

// Присваивание для структуры - копирование.
ss2 = ss1;
```

МОДИФИКАТОРЫ В СТРУКТУРАХ

- new
- public
- protected
- internal
- private

Для типа struct запрещены модификаторы:

- abstract
- sealed
- static

readonly

- Все поля объявлены с модификатором readonly
- Все свойства доступны только для чтения (get и get с init)

ПРИМЕР READONLY СТРУКТУРЫ

```
Point2D first, second, third;
first.X = 10; first.Y = 10;
second.X = 20; second.Y = 20;
third.X = first.X + second.X;
third.Y = first.Y + second.Y;
```

Ошибка компиляции

```
public readonly struct Point2D
{
    public readonly double X { get; init; }
    public readonly double Y { get; init; }
}
```

```
Point2D first = new Point2D { X = 10, Y = 10 },
    second = new Point2D { X = 20, Y = 20 },
    third = new Point2D { X = first.X + second.X, Y = first.Y + second.Y };
```

Описание с инициализацией

```
Console.WriteLine($"First: x = {first.X}, y = {first.Y}");
Console.WriteLine($"Second: x = {second.X}, y = {second.Y}");
Console.WriteLine($"Third: x = {third.X}, y = {third.Y}");
```

ЧЛЕНЫ СТРУКТУР

```
struct_member_declaration
    : constant_declaration
    | field_declaration
    | method_declaration
    | property_declaration
    | event_declaration
    | indexer_declaration
    | operator_declaration
    | constructor_declaration
    | static_constructor_declaration
    | type_declaration
    | fixed_size_buffer_declaration   // unsafe code support
    ;
```

ПОЛЯ

- Могут иметь как ссылочные типы, так и типы значений
- Экземплярные поля структур не допускают инициализации (почему?)

```
public struct Coords
{
    public double x = 0;
    public double y = 0;
}
```

```
public struct Coords
{
    public static readonly int coodInstNumber = 1;

    public double x;
    public double y;
}
```

КОНСТРУКТОРЫ СТРУКТУР

- Явное объявление конструктора без параметров в структуре запрещено до C# 10 (он всегда присутствует неявно)
- Любой конструктор структуры в теле обязан инициализировать все поля
- Явно могут быть описаны конструкторы с параметрами
  - Для них запрещено использование спецификации базы `base(<параметры>)`

```
public struct Coords
{
    public static readonly int coodInstNumber = 1;
    public double x;
    public double y;

    public Coords() { x = 1; y = 1; }
}
```

Допустимо с 10 версии языка

ПРИМЕР. НЕЯВНО СОЗДАННЫЙ КОНСТРУКТОР БЕЗ ПАРАМЕТРОВ

```
Point2D s1 = new();

Point2D s2 = new(5, 10);
Console.WriteLine($"({s1.X}; {s1.Y})");
Console.WriteLine($"({s2.X}; {s2.Y})");

public struct Point2D
{
    public int X;
    public int Y;
    // Конструктор без параметров явно не определён.
    public Point2D(int x, int y) => (X, Y) = (x, y);
}
```

Такой вызов работает. В отличие от классов в структуре всегда есть конструктор умолчания

Вывод:

```
(0; 0)
(5; 10)
```

ЯВНО ОПРЕДЕЛЁННЫЙ КОНСТРУКТОР БЕЗ ПАРАМЕТРОВ (C# 10+)

```
using System;
Point3D s1 = new();
Point3D s2 = new(5, 10);
Point3D s3 = default;
Point3D[] pArr = new Point3D[1];
Console.WriteLine($"({s1.X}; {s1.Y})\n({s2.X}; {s2.Y})");
Console.WriteLine($"({s3.X}; {s3.Y})\n({pArr[0].X}; {pArr[0].Y})");

public struct Point3D {
    public int X;
    public int Y;
    public int Z = 0;                       // OK с C# 10.
    public Point3D() { X = 7; Y = 7; }    // OK с C# 10.
    public Point3D(int x, int y) => (X, Y) = (x, y);
}
```

`default` и массивы игнорируют явно определённый конструктор без параметров.

Вывод:

(7; 7)
(5; 10)
(0; 0)
(0; 0)

DEFAULT

оператор

```
default(тип)
```

```
default(int)  → 0
default(object)  → null
```

литерал

```
default
```

- значение переменной;
- в объявлении значения по умолчанию для необязательного параметра метода
- аргумента при вызове метода
- возврат значения с использованием `return`

ИНИЦИАЛИЗАТОРЫ В КОНСТРУКТОРЕ

```
public struct Person
{
    string name;
    string lastname;
    public string Name { get; init; }
    public string LastName { get; init; }

    public Person(string name, string lastname) => (Name, LastName) = (name, lastname);
}
```

```
public struct Person
{
    public string Name { get; init; }
    public string LastName { get; init; }

    public Person(string name, string lastname) => (Name, LastName) = (name, lastname);
}
```

СТАТИЧЕСКИЙ КОНСТРУКТОР СТРУКТУРЫ

Статистический конструктор не будет автоматически запущен при создании экземпляра структуры с умалчиваемые значениями!

События, запускающие статический конструктор структуры:

- Обращение к статическому члену структуры
- Явный вызов явно описанного конструктора структуры

```
public struct Person
{
    public static int persInstNumb;
    static Person()
    {
        Console.WriteLine("Static");
        persInstNumb = 1;
    }

    public string Name { get; init; }
    public string LastName { get; init; }
    public Person(string name, string lastname) =>
        (Name, LastName) = (name, lastname);
}
```

```
Person p = new Person();
```

Нет вызова статического конструктора

```
Person p2 = new Person("Joe", "Doe");
```

Есть вызов статического конструктора

THIS В СТРУКТУРАХ

```
public struct Point2D
{
    public int X;
    public int Y;
    // Конструктор без параметров явно не определён.
    public Point2D(int x, int y) => (X, Y) = (x, y);
    public Point2D(Point2D p) => this = p;
    public void Change(Point2D p)
    {
        this.X = p.X;
    }
    public override string ToString() => $"({X};{Y})";
}
```

```
Point2D baseP = new Point2D(1, 1);
Point2D newP = new Point2D(baseP);
Console.WriteLine(baseP);
Console.WriteLine(newP);
newP.Y = 15;
newP.Change(baseP);
Console.WriteLine(baseP);
Console.WriteLine(newP);
```

ЧАСТИЧНАЯ ИНИЦИАЛИЗАЦИЯ СТРУКТУР

Допустимо обращение к уже инициализированным полям даже при условии, что не все члены структуры инициализированы:

Вывод:
10

```
Point3D s;
s.X = 10;
Console.WriteLine(s.X);
// Console.WriteLine(s.Y); - ошибка, Y не инициализирована.
// Console.WriteLine(s); - ошибка, s не инициализирована полностью.

public struct Point3D
{
    public int X;
    public int Y;

    public override string ToString() => $"({X}; {Y})";
}
```

РЕАЛИЗАЦИЯ ИНТЕРФЕЙСОВ СТРУКТУРАМИ

- Структура может реализовывать интерфейсы

Исправим

```
public struct Point2D : IComparable<Point2D>
{
    public int X;
    public int Y;
    // Конструктор без параметров явно не определён.
    public Point2D(int x, int y) => (X, Y) = (x, y);
    public Point2D(Point2D p) => this = p;
    public int CompareTo(Point2D other)
    {
        return (X*X) + (Y*Y) > other.X*other.X + other.Y*other.Y?1:-1;
        return 0;
    }
   public override string ToString() => $"({X};{Y})";
}
```

ОТЛИЧИЯ СТРУКТУР И КЛАССОВ

Критерий	**Класс**	**Структура**
Размещение в памяти	Куча	Стек
Производительность	Медленнее	Быстрее
Изменяемость (immutable)	Да	Нет
Наследование	Да	Нет
Инкапсуляция	Private-поля допустимы	Поля всегда public
Конструктор умолчания	Генерируется, если не определен	Не генерируется, имеется неявно
Финализатор	Допустим	Запрещён
Упаковка / распаковка	Не требуется	Требуется

ПЕРЕЧИСЛЕНИЯ

enum

ПЕРЕЧИСЛЕНИЯ

Перечисления – типы значений, членами которых могут являться только именованные целочисленные константы

Синтаксис объявления:

```
[Модификаторы] enum <Идентификатор> [: тип элемента] { [Объявление констант] }
```

ключевое слово

Унаследовать перечисление нельзя

По умолчанию, если не указать явно значения констант, то каждая последующая будет на единицу больше предыдущей. Значение первой константы по умолчанию – ноль

```
public enum Compass
{
    0 South,
    1 North,
    2 West,
    3 East,
}
```

Последняя запятая игнорируется компилятором.

Black	0	Черный цвет.
Blue	9	Синий цвет.
Cyan	11	Голубой цвет (сине-зеленый).
DarkBlue	1	Темно-синий цвет.
DarkCyan	3	Темно-голубой цвет (темный сине-зеленый).
DarkGray	8	Темно-серый цвет.
DarkGreen	2	Темно-зеленый цвет.
DarkMagenta	5	Темно-пурпурный цвет (темный фиолетово-красный).
DarkRed	4	Темно-красный цвет.
DarkYellow	6	Темно-желтый цвет (коричнево-желтый).
Gray	7	Серый цвет.
Green	10	Зеленый цвет.
Magenta	13	Пурпурный цвет (фиолетово-красный).
Red	12	Красный цвет.
White	15	Белый цвет.
Yellow	14	Желтый цвет.

ConsoleColor

Наследование:
Object <- ValueType <- Enum <- ConsoleColor)

ИЗВЕСТНОЕ НАМ ПЕРЕЧИСЛЕНИЕ

ПРИМЕР ОБРАЩЕНИЯ К КОНСТАНТАМ ПЕРЕЧИСЛЕНИЯ

```
using System;
//Вывести на экран строку, символы, которой раскрашены разными цветами консоли.
class Program {
    // метод печатает в консоль символ ch цветом, указанным в параметре color
    public static void PrintColoredChar(char ch, ConsoleColor color)     {
        Console.ForegroundColor = color;
        Console.Write(ch);
        Console.ResetColor();
    }
    static void Main(string[] args)     {
        string testString = "abcdefghijklmnopqrstuvwxyz";
        // в массив строк сохраняем имена цветов консоли
        string[] consoleColors = ConsoleColor.GetNames(typeof(ConsoleColor));
        for (int i = 0; i < testString.Length; i++)
        {
            // печатаем
            PrintColoredChar(testString[i], (ConsoleColor)Enum.Parse(typeof(ConsoleColor),
                consoleColors[i % 16]));
        }
    }
}
```

```
 bcdefghijklmnop rstuvwxyz
```

МОДИФИКАТОРЫ В ПЕРЕЧИСЛЕНИЯХ

- new
- public
- protected
- internal
- private

Для типа enum запрещены модификаторы:

- abstract
- sealed
- static

ПРИМЕР

```
public enum Compass
{
    South,
    North,
    West,
    East
}
```

```
Compass n = Compass.North;
Console.WriteLine(n);
Console.WriteLine((int)n);
```

North

1

```
Console.WriteLine((Compass)2);

int a = 3;
Console.WriteLine((Compass)a);
```

West

East

?

```
Console.WriteLine((Compass)6);
```

ПРИМЕР ПЕРЕЧИСЛЕНИЯ: СВЕТОФОР

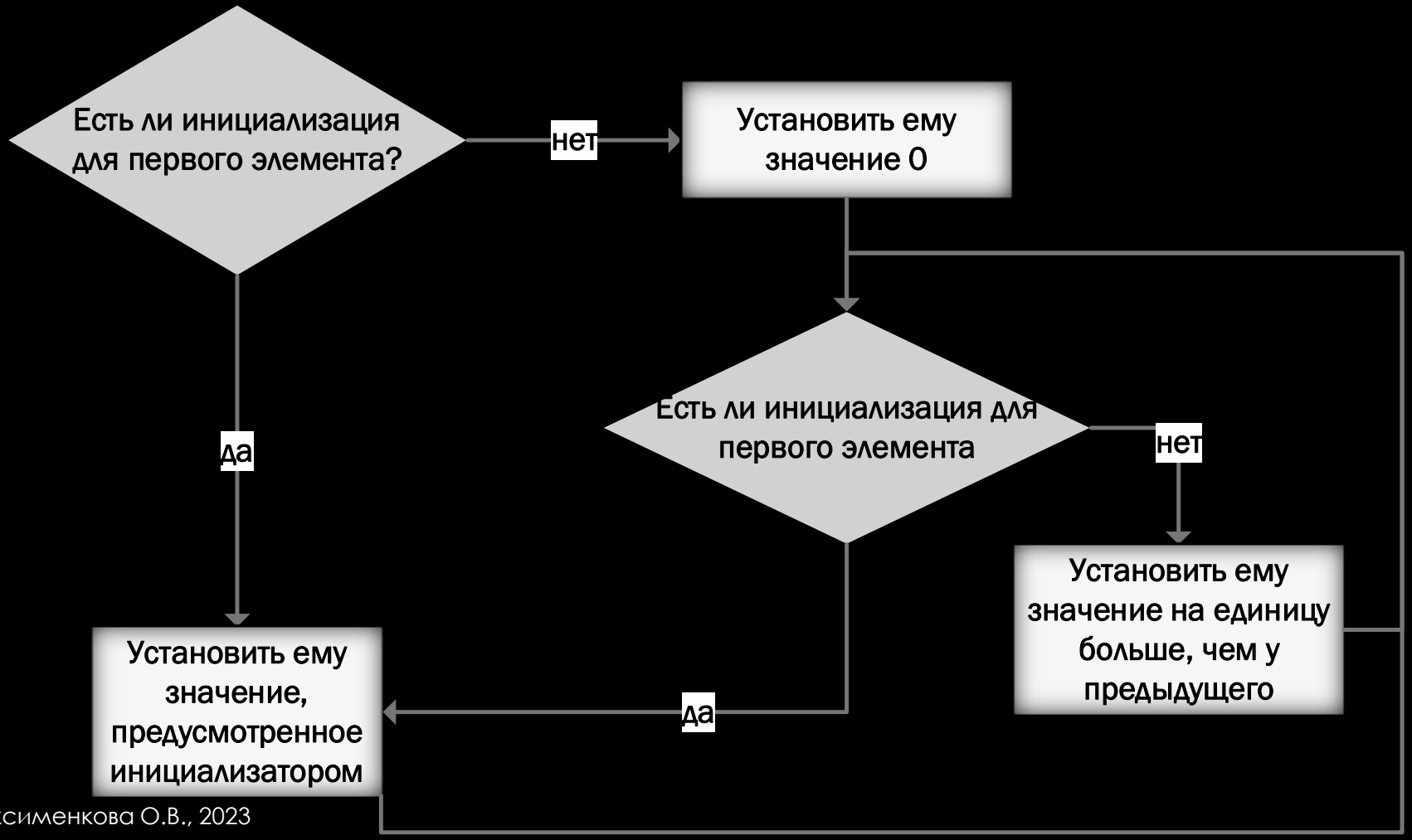
```
public enum TrafficLight
{
    Green,  // 0
    Yellow, // 1
    Red     // 2
}
```

```
TrafficLight t1 = TrafficLight.Green;
TrafficLight t2 = TrafficLight.Yellow;
TrafficLight t3 = TrafficLight.Red;

Console.WriteLine($"{t1}:\t{(int)t1}");
Console.WriteLine($"{t2}:\t{(int)t2}");
Console.WriteLine($"{t3}:\t{(int)t3}");

TrafficLight t4 = t2;// Копирование.
```

Вывод:

```
Green:  0
Yellow: 1
Red:    2
```

ИНИЦИАЛИЗАЦИЯ ЭЛЕМЕНТОВ ПЕРЕЧИСЛЕНИЯ



ПРИМЕРЫ ПЕРЕЧИСЛЕНИЙ

```
public enum CardSuit
{
    Hearts,                 // 0 – первый элемент перечисления (♥).
    Clubs,                  // 1 - +1 к предыдущему (♣).
    Diamonds,               // 2 - +1 к предыдущему (♦).
    Spades,                 // 3 - +1 к предыдущему (♠).
    MaxSuits                // 4 - +1 к предыдущему.
}
```

```
public enum FaceCard : byte
{
    Jack = 11,                      // 11 - Явно заданное значение.
    Queen,                          // 12 - +1 к предыдущему.
    King,                           // 13 - +1 к предыдущему.
    Ace,                            // 14 - +1 к предыдущему.
    NumberOfFaceCards = 4,          // 4 - Явно заданное значение.
    SomeOtherValue,                 // 5 - +1 к предыдущему.
    HighestFaceCard = Ace           // 14 - == Ace (задано выше).
}
```

ПРАВИЛА ИСПОЛЬЗОВАНИЯ ПЕРЕЧИСЛЕНИЙ

Для перечислений после объявления типа с помощью синтаксиса, аналогичного наследованию можно указать тип целочисленных констант – один из встроенных целочисленных типов

- Базовым типом по умолчанию является `int`

```
public enum Compass : byte
{
    South,
    North,
    West,
    East
}
```

```
Console.WriteLine(Enum.IsDefined(typeof(Compass),(Compass)6));
```

Константы перечислений всегда можно явно привести к их базовому типу (и обратно – тоже только явно)

ПРИМЕР

```
public enum Compass : byte
{
    South, Юг = 0,
    North, Север = 1,
    West, Запад = 2,
    East, Восток = 3
}
```

```
byte[] data = { 1, 2, 3, 0, 0, 1, 1, 2, 3, 3, };
foreach (byte b in data) {
    Compass dir = (Compass)b;
    Console.WriteLine(dir switch {
        Compass.South or Compass.Юг => "Backward",
        Compass.North or Compass.Север => "Forward",
        Compass.East or Compass.Восток => "Right",
        Compass.West or Compass.Запад => "Left",
        _ => "No direction"
    });
```

```
Forward
Left
Right
Backward
Backward
Forward
Forward
Left
Right
Right
```

ОПЕРАЦИИ, ОПРЕДЕЛЕННЫЕ ДЛЯ ЗНАЧЕНИЙ ПЕРЕЧИСЛЕНИЙ

- `==`, `!=`, `<`, `>`, `<=`, `>=` (§11.11.6)
- binary `+` (§11.9.5)
- binary `-` (§11.9.6)
- `^`, `&`, `|` (§11.12.3)
- `~` (§11.8.5)
- `++`, `--` (§11.7.14 and §11.8.6)
- `sizeof` (§22.6.9)

```
public enum Compass
{
    South,
    North,
    West,
    East,
}
```

```
public enum TrafficLight
{
    Green,  // 0
    Yellow, // 1
    Red     // 2
}
```

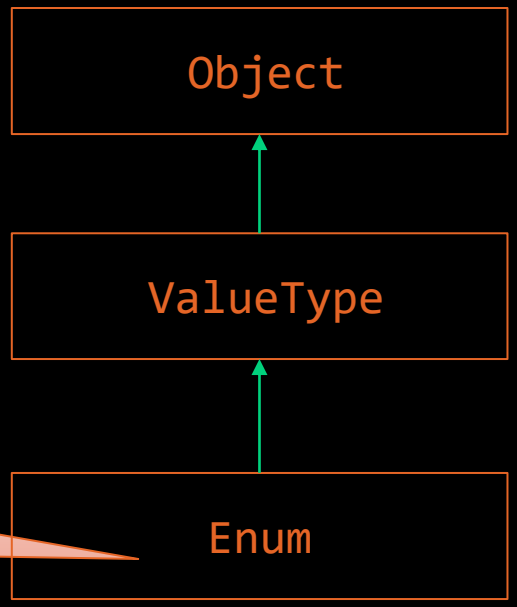
Разные типы данных

```
Console.WriteLine(Compass.West > Compass.South);

Console.WriteLine(Compass.West > TrafficLight.Green);

Console.WriteLine((int)Compass.West > (int)TrafficLight.Green);
```

ТИП SYSTEM.ENUM

- Абстрактный класс базовый для всех типов перечислений
  - Важно: это не базовый тип для констант перечислений!

```
Console.WriteLine("Second element of TrafficLight: {0}",
          Enum.GetName(typeof(TrafficLight), 1));
Console.WriteLine("\nAll elements in TrafficLight:");
foreach (var name in Enum.GetNames(typeof(TrafficLight)))
{
    Console.WriteLine(name);
}
```



Типом перечислений не является

ПЕРЕЧИСЛЕНИЯ И МЕТОДЫ

- Перечисления не могут явно содержать методов
- Допустимо использовать методы расширения

Самостоятельно изучите материалы:

- How to create a new method for an enumeration (C# Programming Guide) (https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/how-to-create-a-new-method-for-an-enumeration)
- Creating An Extension Method To Get Enum Description (https://www.c-sharpcorner.com/article/creating-an-extension-method-to-get-enum-description/)

ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА

- ConsoleColor перечисление (https://learn.microsoft.com/ru-ru/dotnet/api/system.consolecolor?view=net-7.0)
- Типы структур (справочник по c#) (https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/builtin-types/struct)
- Declaration statements (https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/statements/declarations)
  - https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/statements/declarations
- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/reference-types
- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/value-types
- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/enum