

В.В. Подбельский

Использованы иллюстрации пособия Daniel Solis, Illustrated C#

# Иллюстрации к курсу лекций по дисциплине «Программирование на C#»

## Модуль 3. Лекция 4а

### Пользовательские Типы Значений: Структуры и Перечисления

# Ссылочные и Значимые Типы

Типы в C# делятся на ссылочные и значимые в зависимости от области памяти, в которой располагаются их экземпляры.

Ссылочные типы размещаются в куче, а значимые хранятся в стеке.

Пользовательскими *ссылочными типами* являются:

- Классы (class) и записи (record или record class, преобразуются в классы при компиляции);
- Интерфейсы (interface);
- Делегат-типы (delegate).

Пользовательскими *типами значений* являются:

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

# Структуры

**Структуры** – типы значений. Рекомендуется использовать типы структур в качестве легковесных контейнеров для данных, не обладающих сложным поведением. Неявно опечатаны (sealed). Синтаксис объявления структур:

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

## Пример структуры:

ключевое слово



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

## Запрещены модификаторы:

- abstract;
- sealed;
- static;

# Пример Использования Структуры

```
using System;
```

```
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}");
```

## Вывод:

First: x = 10, y = 10

Second: x = 20, y = 20

Third: x = 30, y = 30

```
public struct Point2D
```

```
{
```

```
    public double X;
```

```
    public double Y;
```

```
}
```

Точка представляет собой простой контейнер для двух координат.

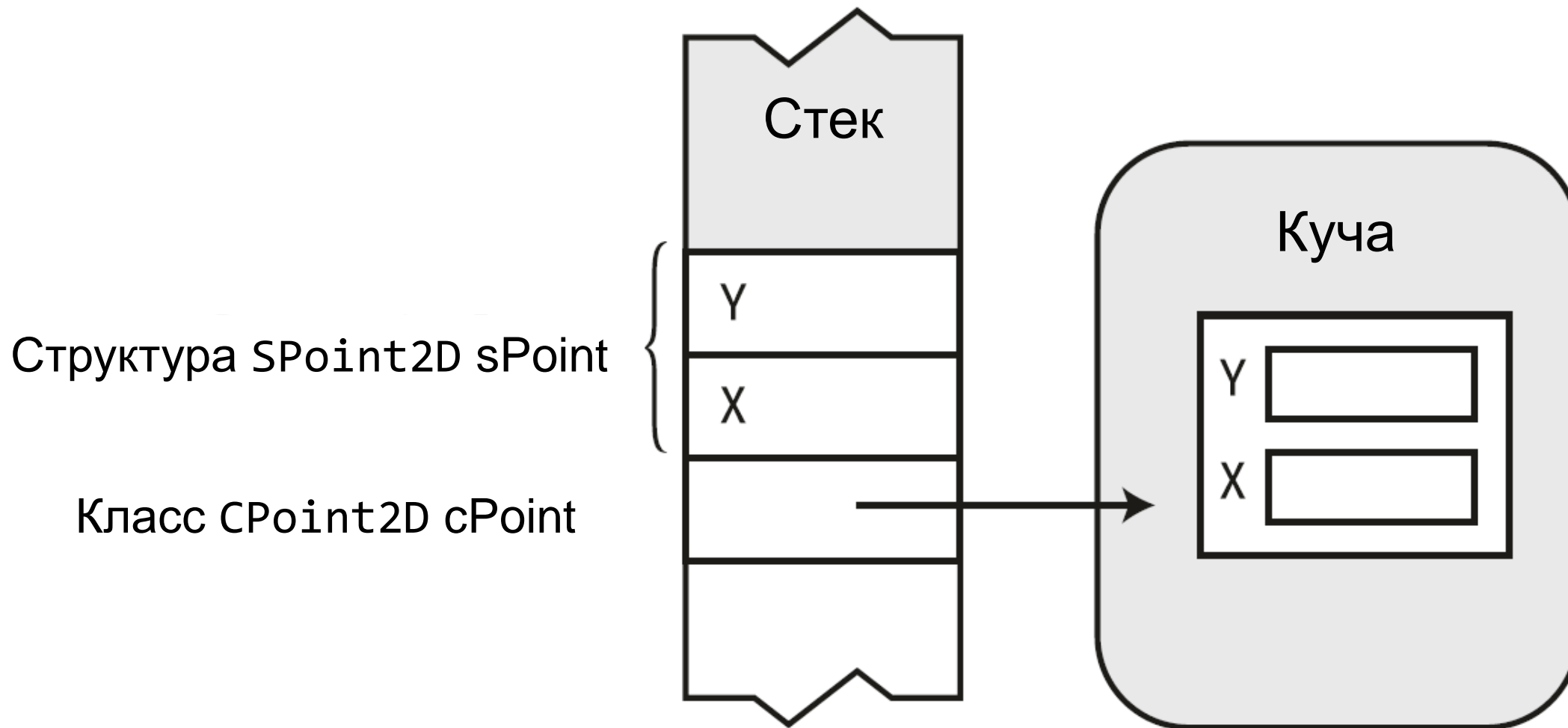
# Сравнение Структуры и Класса в Памяти

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

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

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

# Схема: Структура и Класс в Памяти

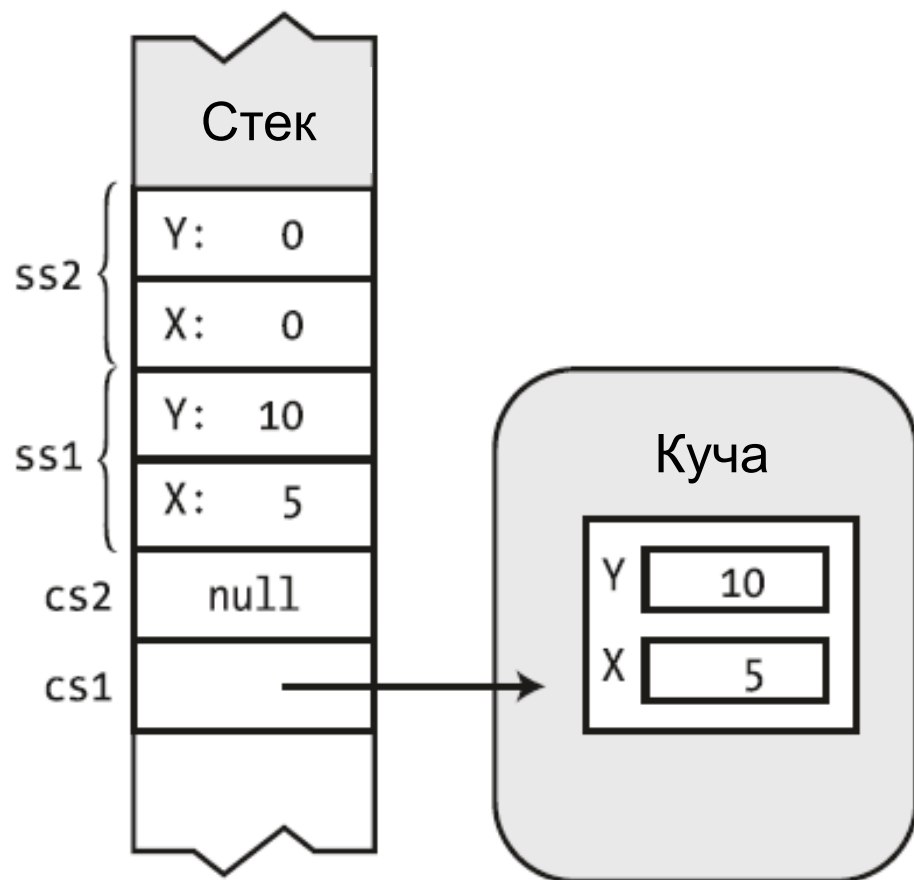


# Присваивание для Структуры и для Класа

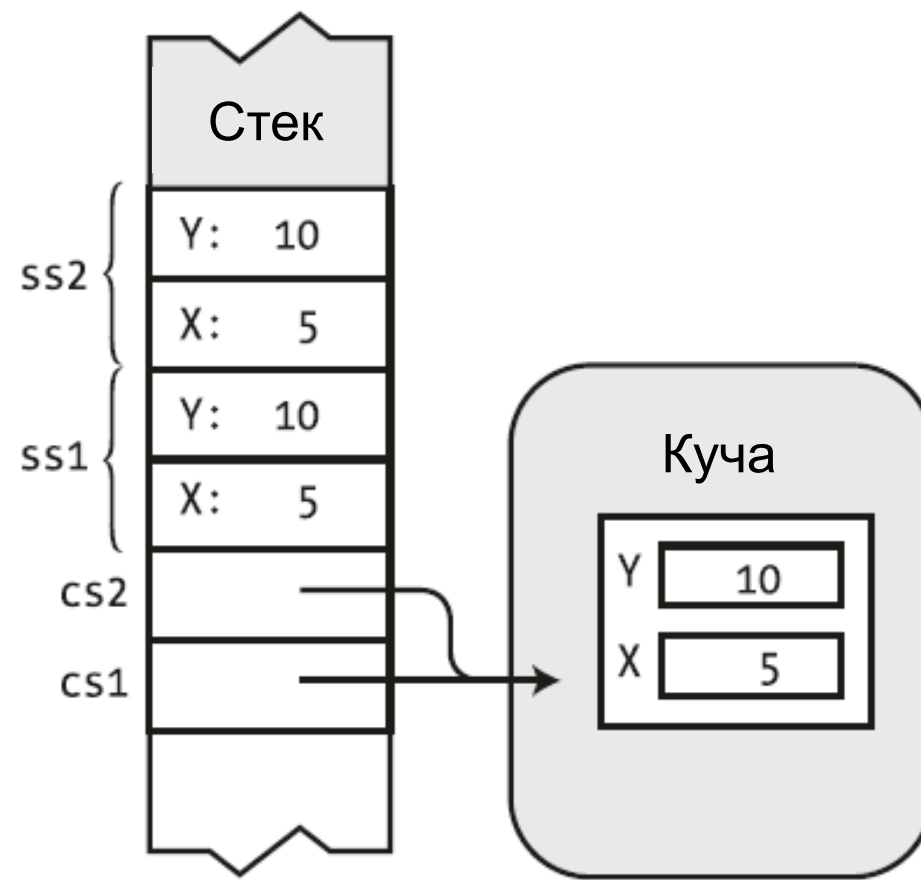
```
CPoint2D cs1 = new CPoint2D(), cs2 = null;  
Point2D ss1 = new Point2D(), ss2 = new Point2D();  
cs1.X = ss1.X = 5;  
cs1.Y = ss1.Y = 10;  
// Присваивание для класса - 2 ссылки на 1 объект.  
cs2 = cs1;  
// Присваивание для структуры - копирование.  
ss2 = ss1;
```

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

# Схема: Присваивание для Структуры и для Класса



До присваивания



После присваивания



# Ограничения Структур. Часть 1

В отличие от классов, структуры имеют ряд ограничений:

- Структуры не наследуются (кроме как от [System.ValueType](#) неявно) и не могут выступать в качестве базовых классов, но могут реализовывать интерфейсы. Как следствие, структуры не могут определять *abstract* и *virtual* члены;
- Структуры не могут быть помечены модификатором *static*;
- У структуры всегда есть конструктор без параметров, который нельзя переопределить. Сгенерированный компилятором конструктор без параметров инициализирует поля значениями по умолчанию (очистка памяти).
- Инициализаторы полей/свойств запрещены.

P.S. Определять *инициализаторы полей/свойств* и конструктор без параметров явно можно только начиная с C# 10.

# Ограничения Структур. Часть 2

В отличие от классов, структуры имеют ряд ограничений:

- Любой конструктор структуры в теле обязан инициализировать все поля;
- Не допускается объявление финализаторов (деструкторов);
- При использовании операции default(T), где T – тип структуры и при создании массива поля структуры **всегда** инициализируются значениями по умолчанию, *даже если конструктор без параметров и(или) инициализаторы полей/свойств были определены явно (C# 10).*

# Генерация Конструктора без Параметров Неявно

```
using System;
```

```
// Конструктор без параметров всё равно доступен.
```

```
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);
```

```
}
```

**Вывод:**

(0; 0)

(5; 10)

# Явно Определённый Конструктор без Параметров

*Warning: данный слайд выходит за рамки темы лекции, т.к. использует 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})");
```

default и массивы игнорируют  
явно определённый  
конструктор без параметров.

```
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);
}
```

## Вывод:

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

# Частичная Инициализация Структур

Для структур доступна частичная инициализация: вы можете обращаться к уже инициализированным полям даже при условии, что не все члены структуры инициализированы:

```
using System;
```

```
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})";
```

```
}
```

**Вывод:**

10

# О модификаторах членов в структурах

Помните, что все структуры неявно опечатаны (sealed) и наследуются от **System.ValueType**.

**Модификаторы, запрещенные для членов структур:**

- **protected**
- **abstract**
- **virtual**

**Допустимые модификаторы для членов структур:**

- **internal**
- **override**
- **new**
- **static**

# Перечисления

**Перечисления** – типы значений (value type), членами которых могут являться только *именованные целочисленные константы*. Синтаксис объявления:

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

## Пример перечисления:

ключевое слово



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

Последняя запятая  
игнорируется компилятором.

# Правила Использования Перечислений

- Для перечислений после объявления типа с помощью синтаксиса, аналогичного наследованию можно указать тип целочисленных констант. Базовым типом по умолчанию является int;
- По умолчанию, если не указать явно значения констант, *то каждая последующая будет на 1 больше предыдущей*. Значение первой константы по умолчанию – 0;
- Константы перечислений всегда можно явно привести к их базовому типу (и обратно – тоже только явно);
- Хотя перечисления не могут явно содержать методов, можно использовать *методы расширения*.



# Пример Перечисления: Светофор

```
using System;
```

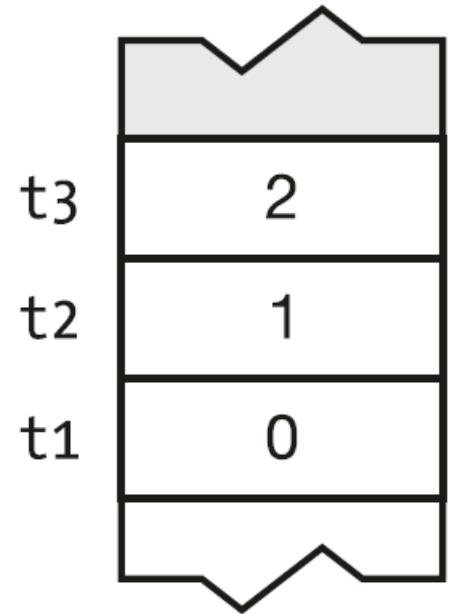
```
public enum TrafficLight  
{  
    Green,    // 0  
    Yellow,   // 1  
    Red       // 2  
}
```

```
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
```



# Явное Указание Типа Констант Перечисления

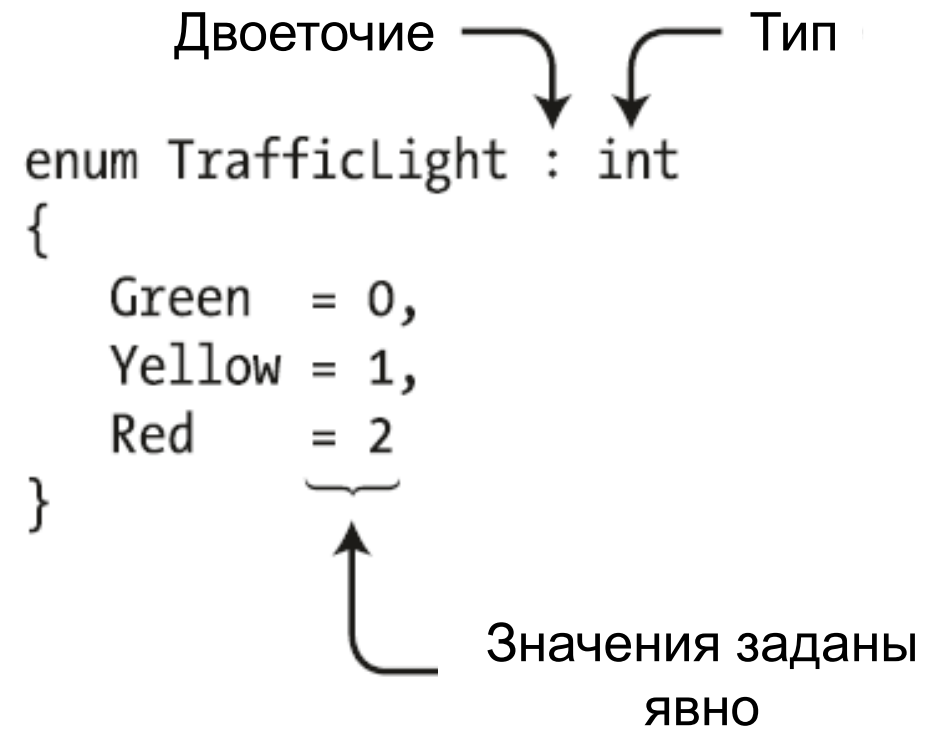
Данные определения эквивалентны:

```
enum TrafficLight
{
    Green,
    Yellow,
    Red
}
```

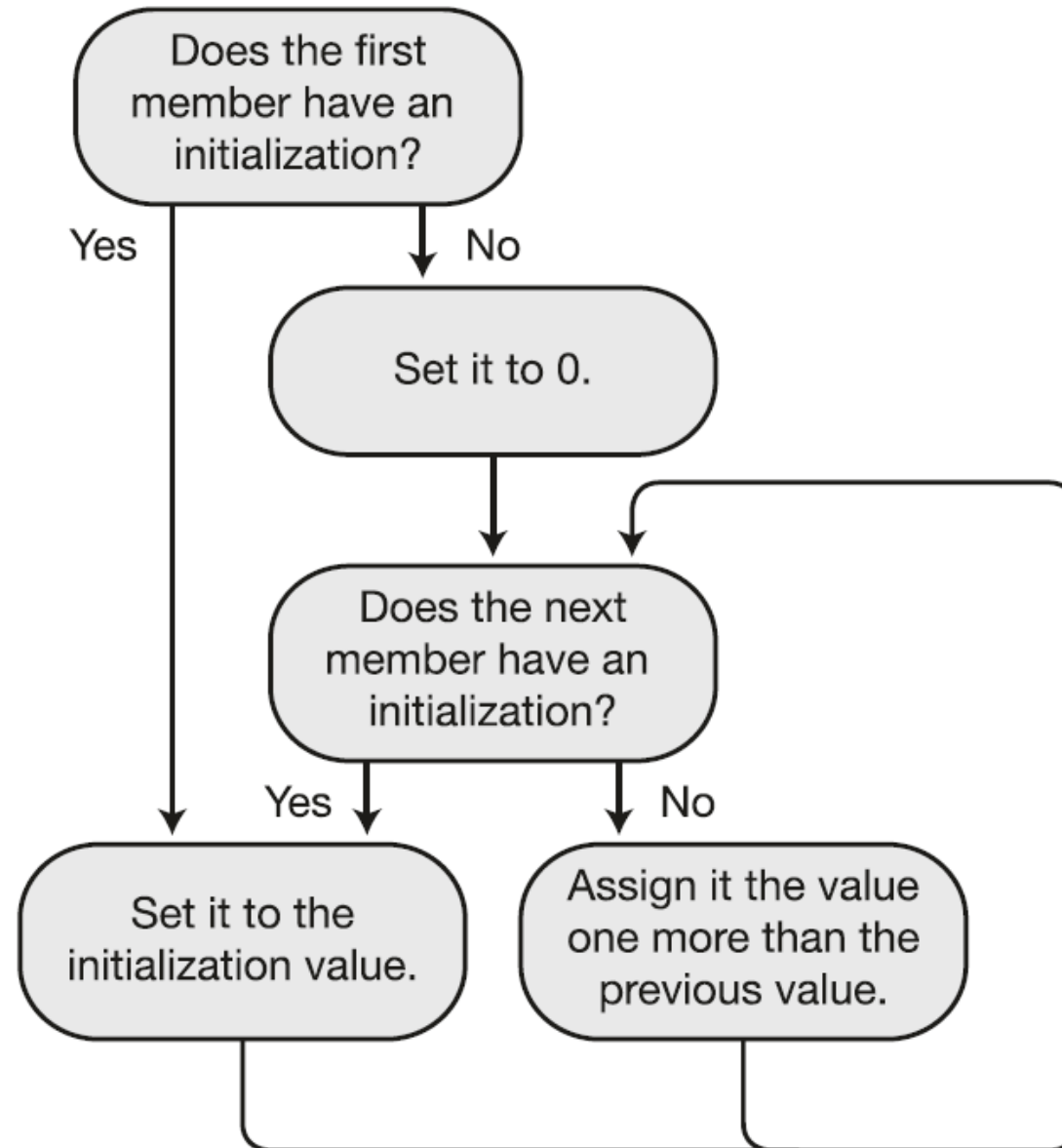
Двоеточие      Тип

```
enum TrafficLight : int
{
    Green    = 0,
    Yellow   = 1,
    Red      = 2
}
```

Значения заданы  
явно



# Инициализация элементов перечисления



# Примеры Перечислений

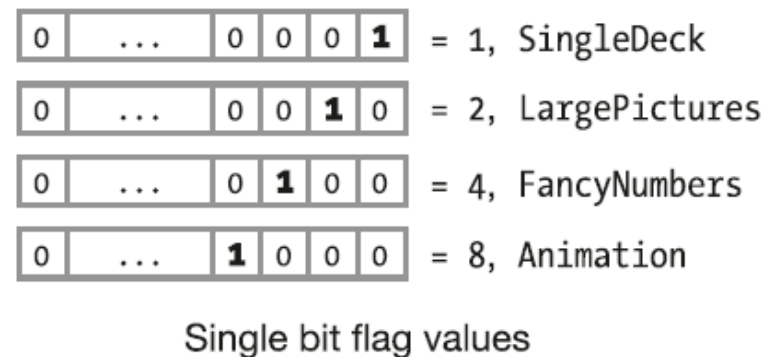
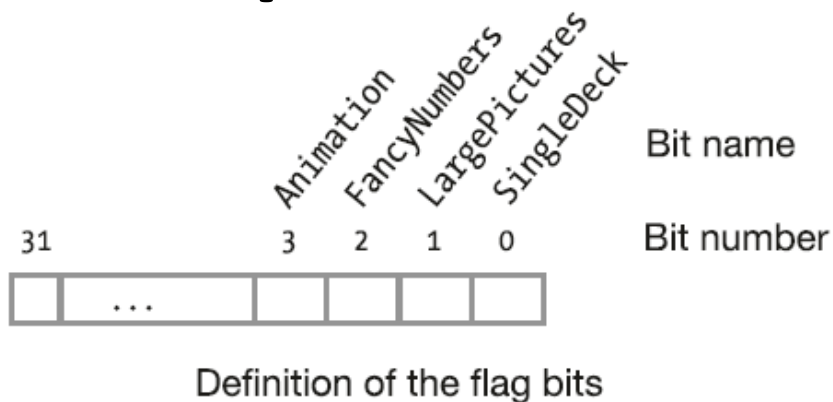
```
public enum CardSuit
{
    Hearts,           // 0 – первый элемент перечисления (♥).
    Clubs,            // 1 - +1 к предыдущему (♣).
    Diamonds,         // 2 - +1 к предыдущему (♦).
    Spades,           // 3 - +1 к предыдущему (♠).
    MaxSuits          // 4 - +1 к предыдущему.
}
```

```
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).
}
```

# Пример Сравнения Элементов Перечислений

```
public enum FirstEnum {  
    Mem1,  
    Mem2  
}  
  
public enum SecondEnum {  
    Mem1,  
    Mem2  
}  
  
if (FirstEnum.Mem1 < FirstEnum.Mem2)                // OK  
    Console.WriteLine("True");  
//if (FirstEnum.Mem1 < SecondEnum.Mem1)              // Ошибка - разные типы  
if ((int)FirstEnum.Mem1 <= (int)SecondEnum.Mem1)     // OK  
    Console.WriteLine("True");
```

# Перечисления как Битовые Флаги

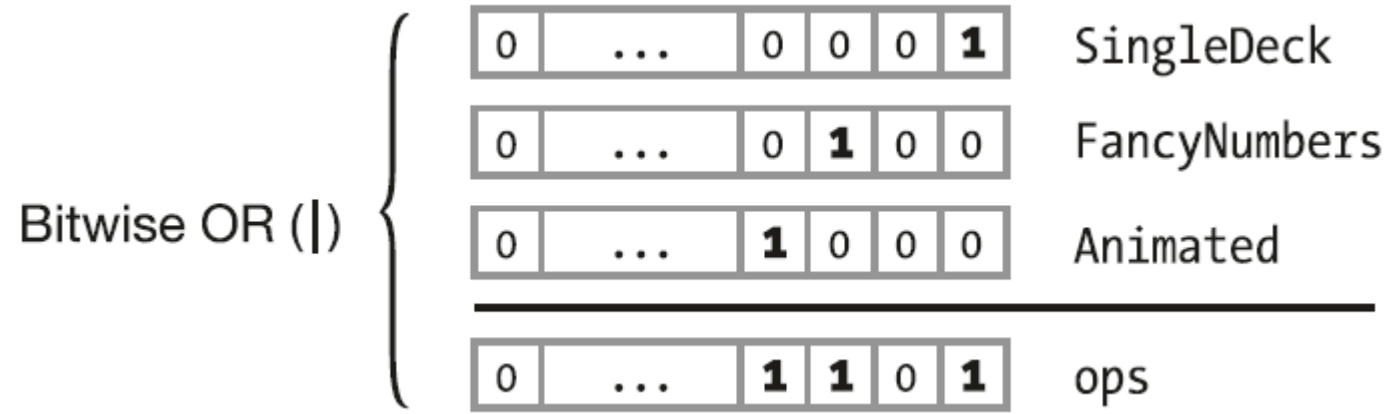


```
// [Flags]
enum CardDeckSettings : uint {
    SingleDeck = 0x01,      // бит 0 (крайний правый)
    LargePictures = 0x02,   // бит 1
    FancyNumbers = 0x04,    // бит 2
    Animation = 0x08        // бит 3
}

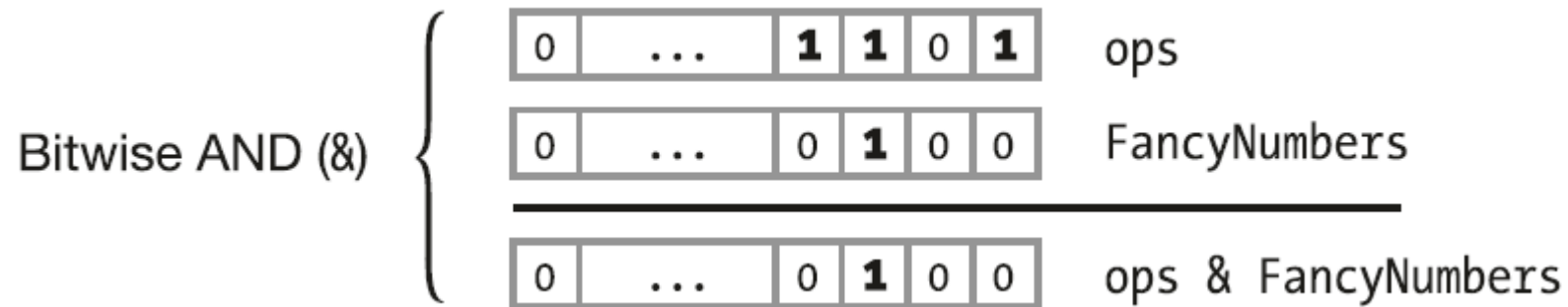
static void Main() {
    CardDeckSettings ops = CardDeckSettings.FancyNumbers;
    Console.WriteLine(ops.ToString());           // FancyNumbers
    // Установка двух разрядов
    ops = CardDeckSettings.FancyNumbers | CardDeckSettings.Animation;
    Console.WriteLine(ops.ToString());           // 12
    Console.WriteLine(ops.HasFlag(CardDeckSettings.Animation)); // True
}
```

# Пример Работы с Флагами

Формирование значения (ops) из нескольких флагов:



Проверка установки флага в значении (ops):



# Использование Атрибута **Flags**

```
[Flags]
enum CardDeckSettings : uint {
    ...
}
```

Метод **ToString()** при выводе значения переменной типа перечисления, помеченного **[Flags]**, выводит имена констант через запятую!

```
CardDeckSettingsF ops = CardDeckSettingsF.FancyNumbers;
Console.WriteLine(ops.ToString());           // FancyNumbers
// Установка двух разрядов:
ops |= CardDeckSettingsF.Animation;
Console.WriteLine(ops.ToString());           // FancyNumbers, Animation
```



# Некоторые Методы System.Enum

Наследуется от **ValueType**.

Реализует интерфейсы: **Comparable**, **Convertible**, **Formattable**.

**Некоторые методы Enum:**

```
public static string GetName (Type enumType, object value);  
    // Возвращает имя константы с заданным значением из перечисления.
```

```
public static string[] GetNames (Type enumType);  
    // Возвращает массив имен констант в перечислении.
```

```
public static Type GetUnderlyingType (Type enumType);  
    // Возвращает базовый тип перечисления.
```

```
public static Array GetValues (Type enumType);  
    // Возвращает массив значений констант в перечислении.
```

```
public bool HasFlag (Enum flag);  
    // Установлены ли одно или несколько битовых полей (this & flag).
```

```
public static bool IsDefined (Type enumType, object value);  
    // Существует ли заданное значение или его имя в виде строки в заданном перечислении.
```

# Пример: GetName() и GetNames()

```
using System;

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);
}

public enum TrafficLight
{
    Green,
    Yellow,
    Red
}
```

## Вывод:

Second element of TrafficLight:  
Yellow

All elements in TrafficLight:  
Green  
Yellow  
Red