

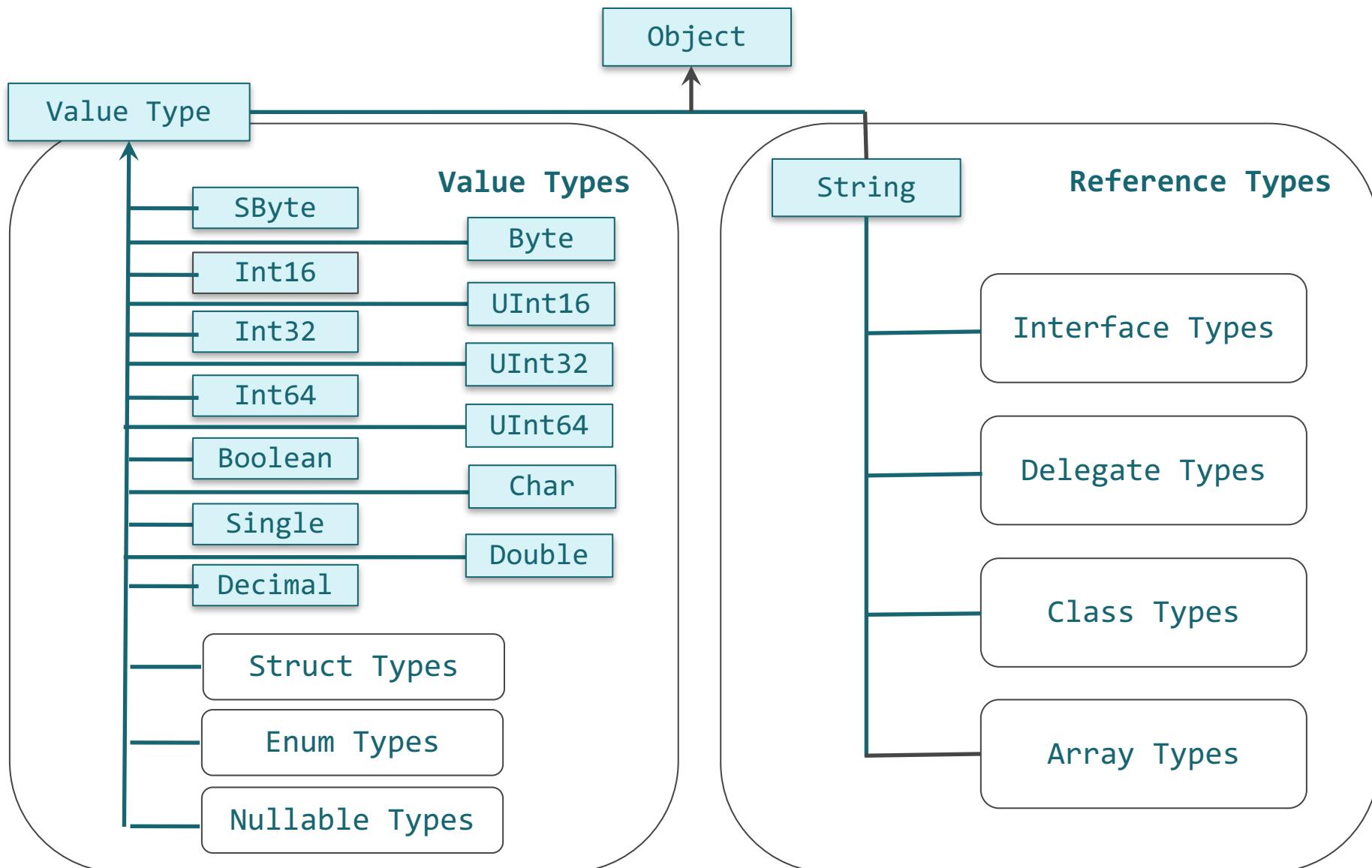


CREATING TYPES IN C#

.NET LAB. MINSK. 2018

Anzhelika KRAVCHUK

Классификация типов



Класс

Понятие класса

Класс – это способ описания сущности, определяющий состояние и поведение, зависящее от этого состояния, а также правила для взаимодействия с данной сущностью (контракт)

```
class ClassName  
{  
    ...  
}
```

Класс определяется с
ключевым словом *class*

С точки зрения программирования класс можно рассматривать как набор данных и методов для работы с ними

С точки зрения структуры программы, класс является сложным типом данных

```
ClassName obj = new ClassName();
```

Объект (экземпляр класса) – это отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом

Понятие класса

[Attributes]

[Class modifiers] class ClassName [Generic type parameters, a base
class, and interfaces]

{

Class members – these are methods, properties, indexers,
events, fields, constructors, overloaded operators,
nested types, and a finalizer

}

public, internal, abstract, sealed, static, unsafe, partial

Класс, объект, ссылка

- **Объект** – это понятие времени выполнения, любой объект является экземпляром класса, создается во время выполнения системы и представляет набор полей
- **Ссылка** - это понятие времени выполнения. Значение ссылки либо null, либо она присоединена к объекту, который она однозначно идентифицирует
- **Сущность** - это статическое понятие (времени компиляции), применяемое к программному тексту, идентификатор в тексте класса, представляющий значение или множество значений в период выполнения. Сущностями являются обычные переменные, именованные константы, аргументы и результаты функций
- Определение ссылки не привязано к аппаратно-программной реализации – «присоединенная» к объекту она может рассматриваться как его абстрактное имя. Отличие ссылки от указателя в ее строгой типизации
- Ссылка в действительности реализована в виде небольшой порции данных, которая содержит информацию, используемую CLR, чтобы точно определить объект, на который ссылается ссылка

Члены класса. Поля

В класс могут добавляться поля и методы, определяющие состояние и поведение класса соответственно

О поле можно думать как о переменной, которая имеет область видимости класс

Статический модификатор	<code>static</code>
Модификатор доступа	<code>public internal private protected</code>
Модификатор наследования	<code>new</code>
Модификатор небезопасного кода	<code>unsafe</code>
Модификатор доступа только для чтения	<code>readonly</code>
Модификатор многопоточности	<code>volatile</code>

Члены класса. Методы

Метод это процедура или функция, определенная внутри класса

Статический модификатор	static
Модификатор доступа	public internal private protected
Модификатор наследования	new virtual abstract override sealed
Модификатор неуправляемого кода	unsafe extern
Модификатор частичного метода	partial
Модификатор асинхронного кода	async

Члены класса. Конструкторы

Метод для инициализации состояния объекта

Статический модификатор	static
Модификатор доступа	public internal private protected
Модификатор неуправляемого кода	unsafe extern

Конструкторы

При определении конструктора соблюдаются следующие правила

- конструкторы имеют то же имя, что и класс, в котором они определены
- конструкторы не имеют типа возвращаемого значения (даже void), но они могут принимать параметры
- конструкторы, как правило, объявляются с модификатором доступа public
- конструкторы обычно инициализируют некоторые или все поля объекта, а также могут выполнять любые дополнительные задачи инициализации, требуемые классу

```
public class Residence
{
    public Residence(ResidenceType type, int numberOfBedrooms) { }
    public Residence(ResidenceType type, int numberOfBedrooms,
                    bool hasGarage) { }
    public Residence(ResidenceType type, int numberOfBedrooms,
                    bool hasGarage, bool hasGarden) { }
}
```

CLR вызывает конструкторы автоматически!

Конструкторы

```
public class Residence
{
    private ResidenceType type;
    private int numberOfBedrooms;
    private bool hasGarage;
    private bool hasGarden;
    private Residence(ResidenceType type, int numberOfBedrooms, bool
hasGarage, bool hasGarden)
    {
        this.type = type;
        this.numberOfBedrooms = numberOfBedrooms;
        this.hasGarage = hasGarage;
        this.hasGarden = hasGarden;
    }
    public Residence() : this(ResidenceType.House, 3, true, true{ }
    ...
}
```

Использование this в классе

- Когда this используется в выражении внутри экземплярного конструктора класса, оно классифицируется как значение, типом которого является тип экземпляра класса, в которой оно используется и это значение является ссылкой на сконструированный объект.
- Когда this используется в выражении внутри метода экземпляра или кода доступа к экземпляру класса, оно классифицируется как значение. Типом значения является тип экземпляра класса, внутри которого оно используется и это значение является ссылкой на объект, для которого вызывается метод или код доступа.

Создание объектов

```
public class Employee
{
    private int id;           ← ..... Экземплярные поля
    private string name;
    private static CompanyPolicy policy; ← ..... статическое поле

    public virtual void Work()
    {
        Console.WriteLine("Zzzz...");
    }

    public void TakeVacation(int days)
    {
        Console.WriteLine("Zzzz...");
    }

    public static void SetCompanyPolicy(CompanyPolicy plc)
    {
        policy = plc;
    }
}
```

← Экземплярный виртуальный метод

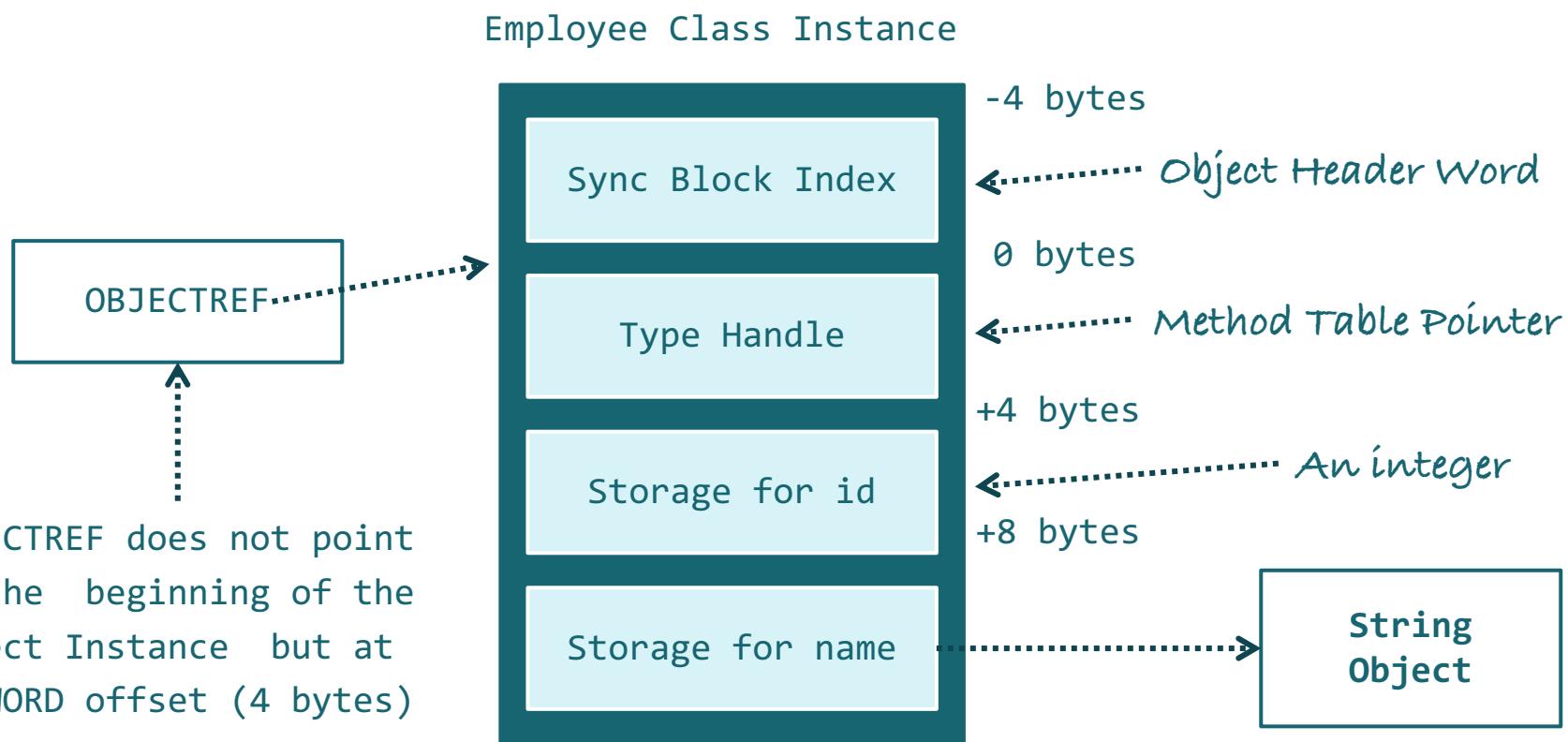
← Экземплярный метод

← статический метод

Создание объектов

- ЕЕ (Execution Engine) выделяет память под объект
- ЕЕ инициализирует указатель на таблицу методов - фактически после этого этапа объект является полноценным живым объектом
- ЕЕ закладывает указатель на объект в регистр `esx` и передает управление конструктору, указанному в инструкции `newobj`, породившей генерацию кода создания объекта
- Если во время работы конструктора не произошло необработанных исключений, то ссылка на объект помещается в ту или иную переменную области видимости, из которой вызывался код создания объектов

Создание объектов



Разделяемые классы и разделяемые методы

Частичные типы позволяют разделить определение типа на несколько файлов. Обычный сценарий заключается в том, что частичный класс должен быть автоматически сгенерирован из какого-либо другого источника (такого как шаблон или дизайнер Visual Studio), и этот класс должен быть дополнен дополнительными методами

```
// PaymentFormGen.cs - auto-generated  
partial class PaymentForm { ... }
```

```
// PaymentForm.cs - hand-authored  
partial class PaymentForm { ... }
```

```
// PaymentFormGen.cs - auto-generated  
partial class PaymentForm { ... }
```

```
// PaymentForm.cs - hand-authored  
class PaymentForm { ... }
```



Частичные типы полностью разрешаются компилятором, что означает, что каждый участник должен быть доступен во время компиляции и должен находиться в одной и той же сборке.

Использование разделяемых классов и разделяемых методов

Частичный тип может содержать частичные методы. Они позволяют автогенерируемому частичному типу предоставлять настраиваемое поведение

```
partial class PaymentForm // In auto-generated file
{ ...
    partial void ValidatePayment (ref decimal amount);
}
```

←..... *Определение*

```
partial class PaymentForm // In hand-authored file
{ ...
    partial void ValidatePayment (ref decimal amount)
    {
        if(amount > 100) ...
    }
}
```

←..... *реализация*

неявно приватный

Структура

Понятие структуры

Структура похожа на класс со следующими ключевыми отличиями:

- структура - это тип значения, тогда как класс является ссылочным типом
- структура не поддерживает наследование (отличное от неявно полученного System.Object, точнее, System.ValueType)

Структура может иметь все члены класса, кроме следующих:

- конструктор без параметров
- инициализаторы не статических полей
- финализатор
- виртуальные или защищенные члены

Поскольку структура является типом значения, каждый экземпляр, является переменной соответствующего типа (переменная сама хранит свое значение), не требует дополнительных байт в памяти; это приводит к полезной экономии при создании многих экземпляров структурного типа. Например, для создания массива типа значения требуется только одно выделение памяти в кучи.

Определение и использование структуры

```
struct Currency
{
    public string currencyCode; // The ISO 4217 currency code
    public string currencySymbol; // The currency symbol ($,£,...)
    public int fractionDigits; // The number of decimal places
}
```

Чтобы создать экземпляр структуры, нет необходимости использовать оператор new, однако структура в этом случае считается неинициализированной

```
Currency unitedStatesCurrency;
unitedStatesCurrency.currencyCode = "USD";
unitedStatesCurrency.currencySymbol = "$";
unitedStatesCurrency.fractionDigits = 2;
```

Способы
инициализации
структур

```
Currency unitedStatesCurrency = new Currency();
Currency unitedStatesCurrency = default(Currency);
```

```
IL_0001: ldloca.s 00 // unitedStatesCurrency
IL_0003: initobj UserQuery.Currency
IL_0009: ldloca.s 01 // unitedStatesCurrency
IL_000B: initobj UserQuery.Currency
```

Определение и использование структуры. Использование this

- Когда `this` используется в выражении внутри экземплярного конструктора структуры, оно классифицируется как переменная, типом которой является тип экземпляра структуры, в которой оно используется. Эта переменная представляет создаваемую структуру. Переменная `this` конструктора экземпляра ведет себя в точности так же, как параметр `out` структурного типа, в частности, это означает, что переменная должна быть определено присвоена в каждом пути выполнения конструктора экземпляра.
- Когда `this` используется в выражении внутри метода экземпляра или кода доступа к экземпляру структуры, оно классифицируется как переменная. Типом переменной является тип экземпляра структуры, внутри которой он используется. Если метод или код доступа не является итератором, переменная `this` представляет структуру, для которой вызывается метод или код доступа, и ведет себя в точности так же, как параметр `ref` структурного типа. Если метод или код доступа является итератором, переменная `this` представляет копию структуры, для которой вызывается метод или код доступа, и ее поведение в точности такое же, как у параметра-значения значимого (структурного) типа.

Инициализация структуры

```
struct Currency
{
    public string currencyCode;      // The ISO 4217 currency code.
    public string currencySymbol;    // The currency symbol ($,£,...).
    public int fractionDigits = 2;   // The number of decimal places.

    public Currency(string code, string symbol)
    {
        this.currencyCode = code;
        this.currencySymbol = symbol;
        this.fractionDigits = 2;
    }
};

...
Currency unitedKingdomCurrency = new Currency("GBP", "£");
```



Правила обязательной инициализации
всех полей структуры, аналогичные
правилам для локальных переменных
(definite assignment rules)

Инициализация структуры

```
struct Currency
{
    public string currencyCode;
    public string currencySymbol;
    public int fractionDigits;

    public Currency(string code, string symbol) : this()
    {
        fractionDigits = 2;
    }
};
```

Вызов `this()` превращается в инструкцию `initobj`, используемую для получения значения по умолчанию экземпляра структуры



```
public Currency(string code, string symbol) : this()
```



Поля `currencyCode` и `currencySymbol` инициализированы неявно!

Определение и использование структуры

Данные в переменных структурного типа хранятся своим значением

Структуры используются для моделирования элементов, которые содержат относительно небольшое количество данных

Для структурных типов нельзя использовать по умолчанию многие из общих операций, таких как == и !=, если для них не предоставлены перегрузки этих операций

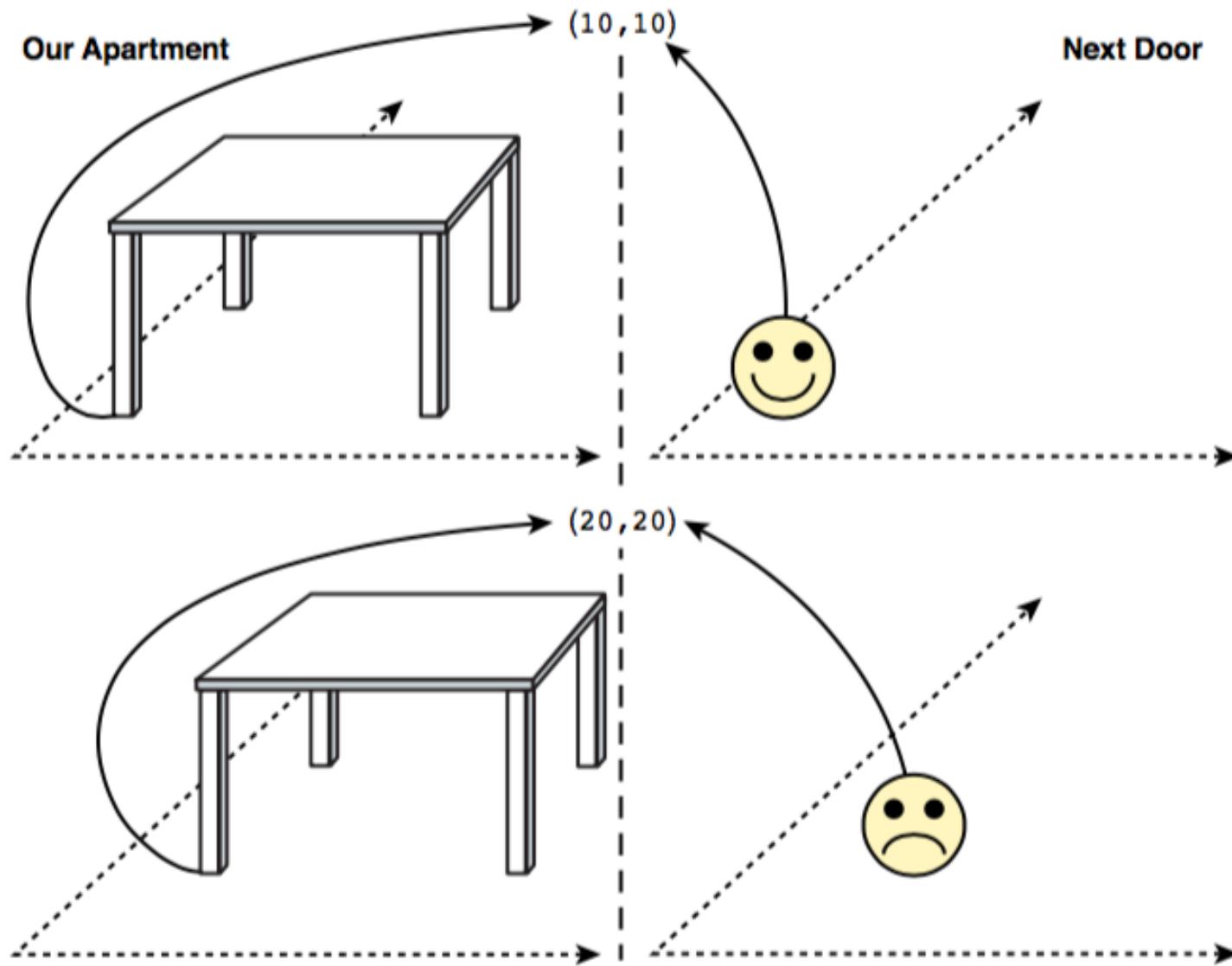
Классы vs структуры. Point - class

```
class Kid
{
    public Point Location { get; set; }
}

class Table
{
    public Point Location { get; set; }
}

class Point
{
    public Point(int x, int y)
    {
        this.X = x;
        this.Y = y;
    }
    public int X { get; set; }
    public int Y { get; set; }
}
```

Классы vs структуры. Point - class



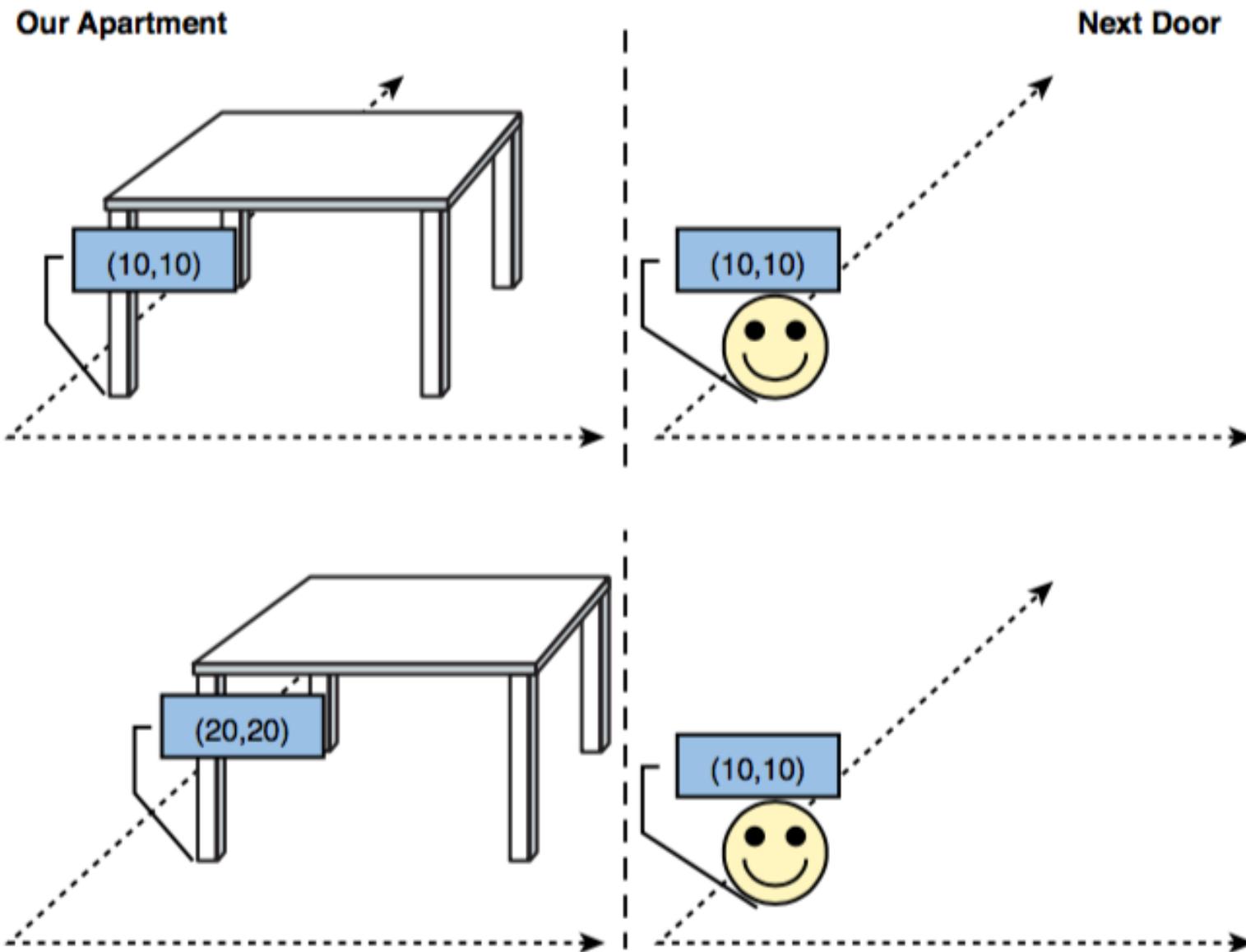
Классы vs структуры. Point - struct

```
class Kid
{
    public Point Location { get; set; }
}

class Table
{
    public Point Location { get; set; }
}

struct Point
{
    public Point(int x, int y)
    {
        this.X = x;
        this.Y = y;
    }
    public int X { get; set; }
    public int Y { get; set; }
}
```

Классы vs структуры. Point - struct



Перечисление

Перечисление

d = 5;

vs

d = DayOfWeek.Friday;



Код легче поддерживать, поскольку определяются только ожидаемые значения переменных

Код легче читать, потому что присваиваются легко идентифицированные имена

Код легче в наборе, поскольку IntelliSense выводит список возможных значений, которые можно использовать

Перечислимые типы подвергаются строгой проверке типов

Перечисление

Каждый перечислимый тип прямо наследует System.Enum, производному от System.ValueType, а тот в свою очередь — System.Object

Перечислимые типы относятся к значимым типам и могут выступать как в неупакованной, так и в упакованной формах

Перечисления создаются с помощью ключевого слова enum

```
public enum Color
{
    White,
    Red,
    Green,
    Blue,
    Orange
}
```

//psevdocode

```
public struct Color : System.Enum
{
    public const Color White = (Color) 0;
    public const Color Red = (Color) 1;
    public const Color Green = (Color) 2;
    public const Color Blue = (Color) 3;
    public const Color Orange = (Color)
4;

    public Int32 value__;
}
```

Перечисление

E Color

- ▶ .class enum nested private auto ansi sealed
- ▶ extends [mscorlib]System.Enum
- ↳ S Blue : public static literal valuetype Enums.Enums/Color
- ↳ S Green : public static literal valuetype Enums.Enums/Color
- ↳ S Orange : public static literal valuetype Enums.Enums/Color
- ↳ S Red : public static literal valuetype Enums.Enums/Color
- ↳ S White : public static literal valuetype Enums.Enums/Color
- ↳ D value__ : public specialname rtspecialname int32

Color:Orange : public static literal valuetype Enums.Enums/Color

Find	Find Next	X
------	-----------	---

```
.field public static literal valuetype Enums.Enums/Color Orange = int32(0x00000004)
```

Color:Green : public static literal valuetype Enums.Enums/Color

Find	Find Next	X
------	-----------	---

```
.field public static literal valuetype Enums.Enums/Color Green = int32(0x00000002)
```

Создание новых типов перечисления

Перечисления можно объявить в классе или пространстве имен, но нельзя в методе

```
enum Season
{
    Spring,           enum Season          Базовый класс FCL (System.Int32)
    Summer,
    Autumn,
    Winter
};

enum Season
{
    Spring = 1,       enum Season
    Summer,          {
    Autumn,          Spring = 1,
    Winter           Summer,
                    Autumn = 3,
                    Fall = 3,
                    Winter
    };
};

enum Season : short
{
    Spring,
    Summer,
    Autumn,
    Winter
};
```

byte sbyte short ushort int uint long ulong

Инициализация и присваивание переменных перечисления

Объявление переменных перечисления и присваивание им значений выполняется аналогично другим типам в C#

```
enum Day
{
    Monday = 1,
    Tuesday = 2,
    Wednesday = 3,
    Thursday = 4,
    Friday = 5,
    Saturday = 6,
    Sunday = 7
};
```

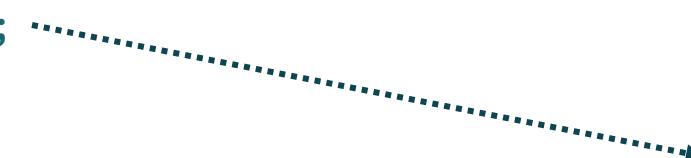
```
static void Main(string[] args)
{
    Day dayOff = Day.Sunday;
}
```

[EnumType] variableName = [EnumValue]

Инициализация и присваивание переменных перечисления

С переменными типа перечисления можно выполнять простые операции во многом таким же образом, как и с переменными целого типа

```
for(Day dayOfWeek = Day.Monday; dayOfWeek <= Day.Sunday; dayOfWeek++)  
{  
    Console.WriteLine(dayOfWeek);  
}
```



Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday

Переменные перечисления можно сравнивать

Для переменных перечисления можно выполнять целочисленные операции, такие как инкремент и декримент

«==», «!=», «<», «>», «<=», «>=»

Day.Monday + Day.Wednesday



Упаковка и распаковка

```
Residence house = new Residence(...); ← ..... Класс  
object obj = house;
```

```
Currency currency = new Currency(...); ← ..... структура  
object o = currency; ← ..... Упаковка
```

Упаковка (внутренне)

Выделяется память в управляемой куче. Объем выделенной памяти определяется размером, требуемый полям типа значения, и двумя дополнительными членами служебной информации (указатель на объект тип и индекс блока синхронизации), требуемый всем объектам в управляемой куче.

Поля типа значения копируются в выделенную память в куче.

Возвращается адрес созданного объекта. Этот адрес теперь является ссылкой на объект - тип значения теперь является ссылочным типом.

Упаковка и распаковка

```
Currency currency = new Currency(...);  
object o = currency;  
...  
Currency anotherCurrency = (Currency)o;
```

Для получения значения упакованной копии необходимо использовать операцию приведения типов

Распаковка (внутренне)

CLR проверяет тип объекта

Если типы совпадают, извлекает значение из упакованного объекта в куче и копирует его в переменную в стеке

Упаковка и распаковка

```
static void Main()
{
    Bar(42);
}
static void Bar(object value)
{
    int a = (int)value;
}
```

IL_0000:	nop	
IL_0001:	ldc.i4.s	2A
IL_0003:	box	System.Int32
IL_0008:	call	UserQuery.Bar
IL_000D:	nop	
IL_000E:	ret	

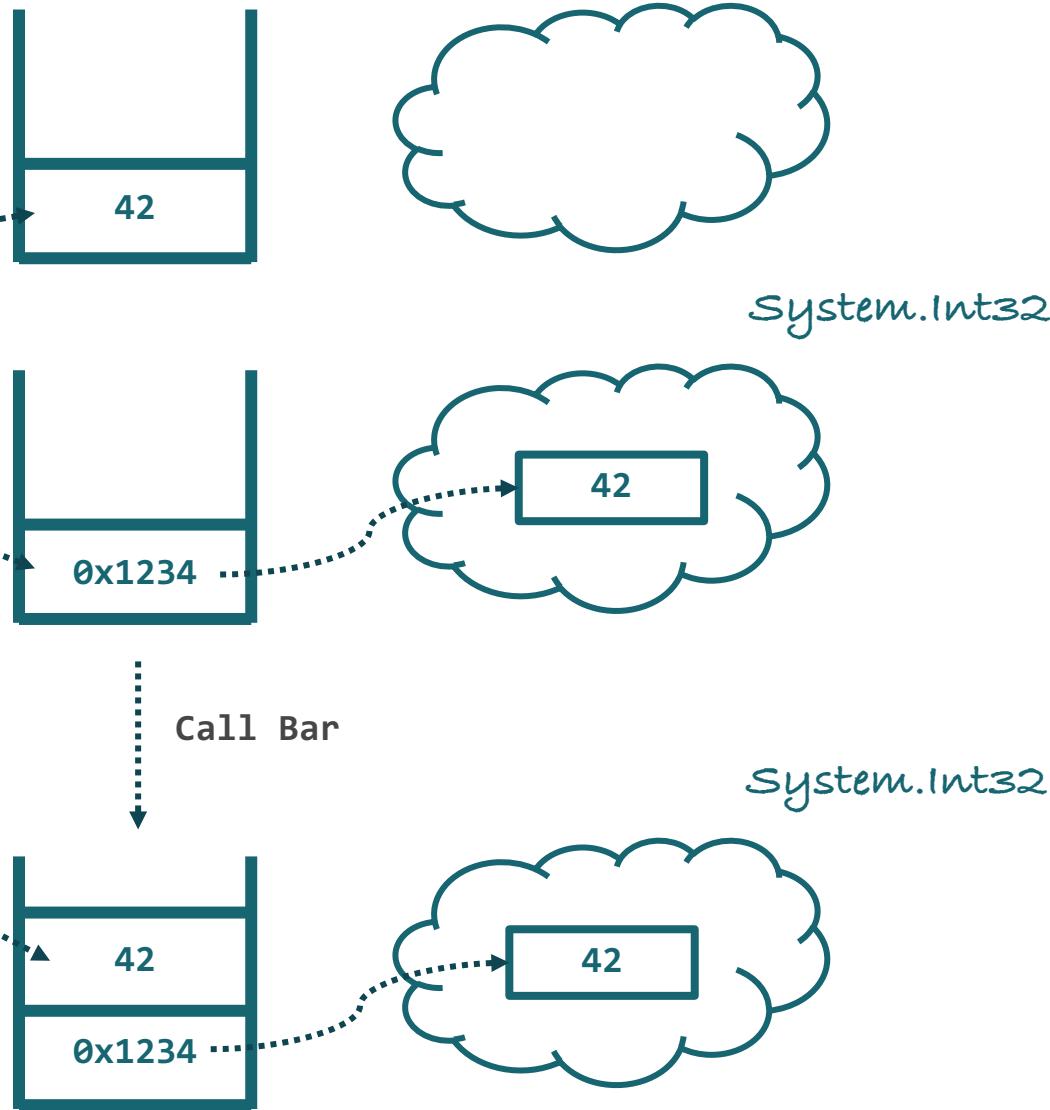
Bar:

IL_0000:	nop	
IL_0001:	ldarg.0	
IL_0002:	unbox.any	System.Int32
IL_0007:	stloc.0	// a
IL_0008:	ret	

Упаковка и распаковка

```
IL_0000: nop  
IL_0001: ldc.i4.s 2A  
IL_0003: box System.Int32  
IL_0008: call UserQuery.Bar  
IL_000D: nop  
IL_000E: ret
```

```
Bar:  
IL_0000: nop  
IL_0001: ldarg.0  
IL_0002: unbox.any System.Int32  
IL_0007: stloc.0 // a  
IL_0008: ret
```



Обнуляемые типы

При объявлении ссылочной переменной можно установить ее значение в null, чтобы указать, что она не инициализирована

```
Residence house = null;  
...  
if (house == null)  
{  
    house = new Residence(...);  
}
```

```
Currency currency = null;
```



структура –
тип значения

Чтобы указать, что тип значения является обнуляемым, используется знак вопроса «?»

```
Currency? currency = null;  
...  
if (currency == null)  
{  
    currency = new Currency(...);  
}
```

Обнуляемые типы

Типы, допускающие значения null, по сути являются экземплярами структуры System.Nullable<T>

Nullable<Int32> - любое значение от -2 147 483 648 до 2 147 483 647
или значение null

Nullable<bool> - значения true, false или null

```
Currency? currency = null;  
...  
if (currency.HasValue)  
{  
    Console.WriteLine(currency.Value);  
}
```

Указывает, содержит ли обнуляемый тип значение или null

Содержит значение переменной
(только для чтения)

Обнуляемые типы

```
int? i = null;  
int j = 99;  
i = 100  
i = j;  
j = i;
```

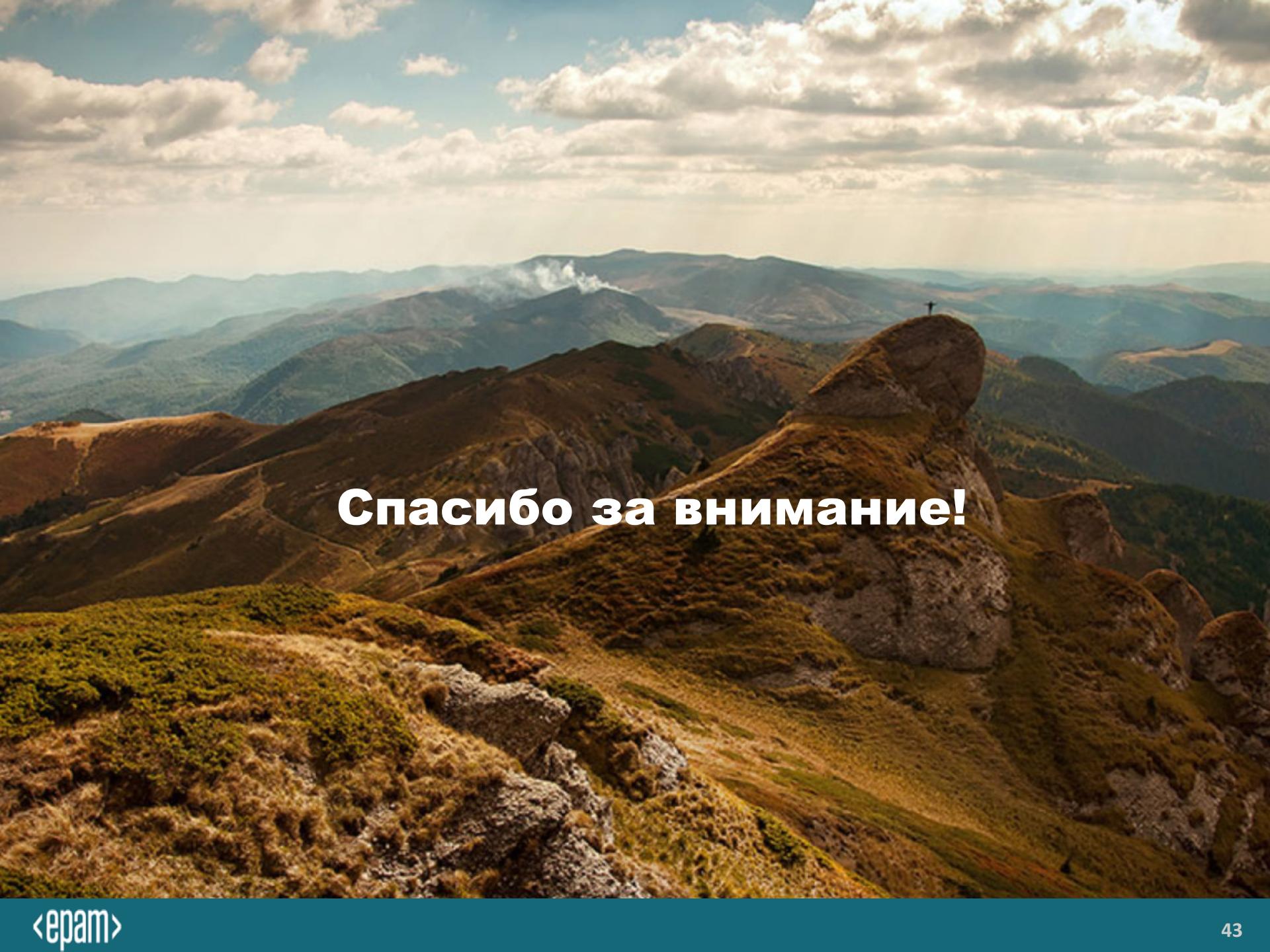
Операция поглощения (null-coalescing) «??» используется для определения значения по умолчанию для обнуляемых значимых и ссылочных типов. Он возвращает левый операнд, если он не является нулевым, в противном случае он возвращает правый.

```
int x = (b.HasValue) ? b.Value : 123;      vs      int x = b ?? 123;
```

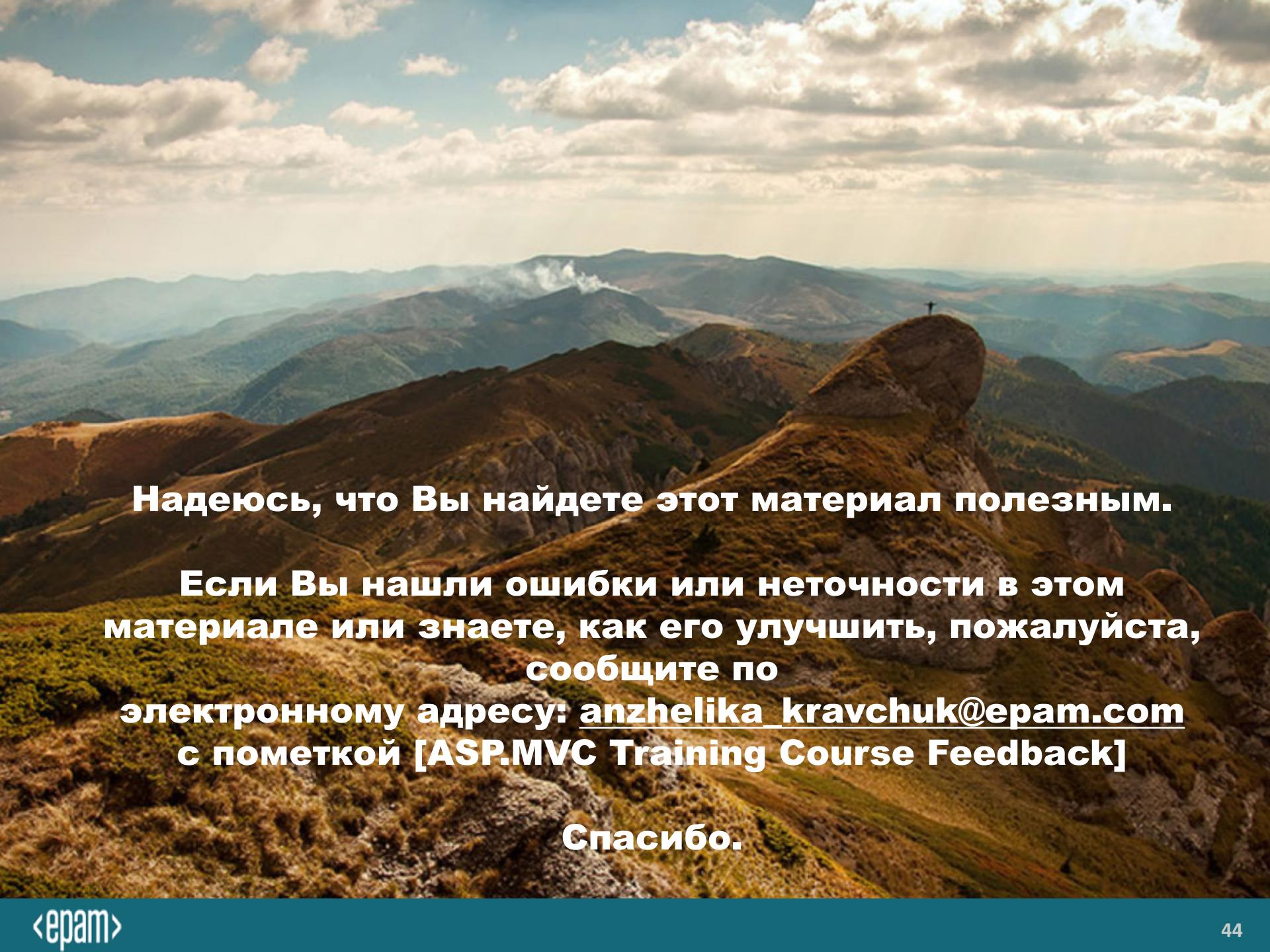
```
string filename = (GetFilename() != null) ? temp : "Untitled";
```

vs

```
string filename = GetFilename() ?? "Untitled";
```

A wide-angle photograph of a mountain range under a dramatic sky. In the foreground, rocky terrain and green slopes are visible. In the middle ground, a person stands on a prominent, rounded mountain peak. The background features multiple layers of mountains, with a plume of white smoke or steam rising from the top of one in the distance.

Спасибо за внимание!



Надеюсь, что Вы найдете этот материал полезным.

**Если Вы нашли ошибки или неточности в этом
материале или знаете, как его улучшить, пожалуйста,
сообщите по**

**электронному адресу: anzhelika_kravchuk@epam.com
с пометкой [ASP.MVC Training Course Feedback]**

Спасибо.