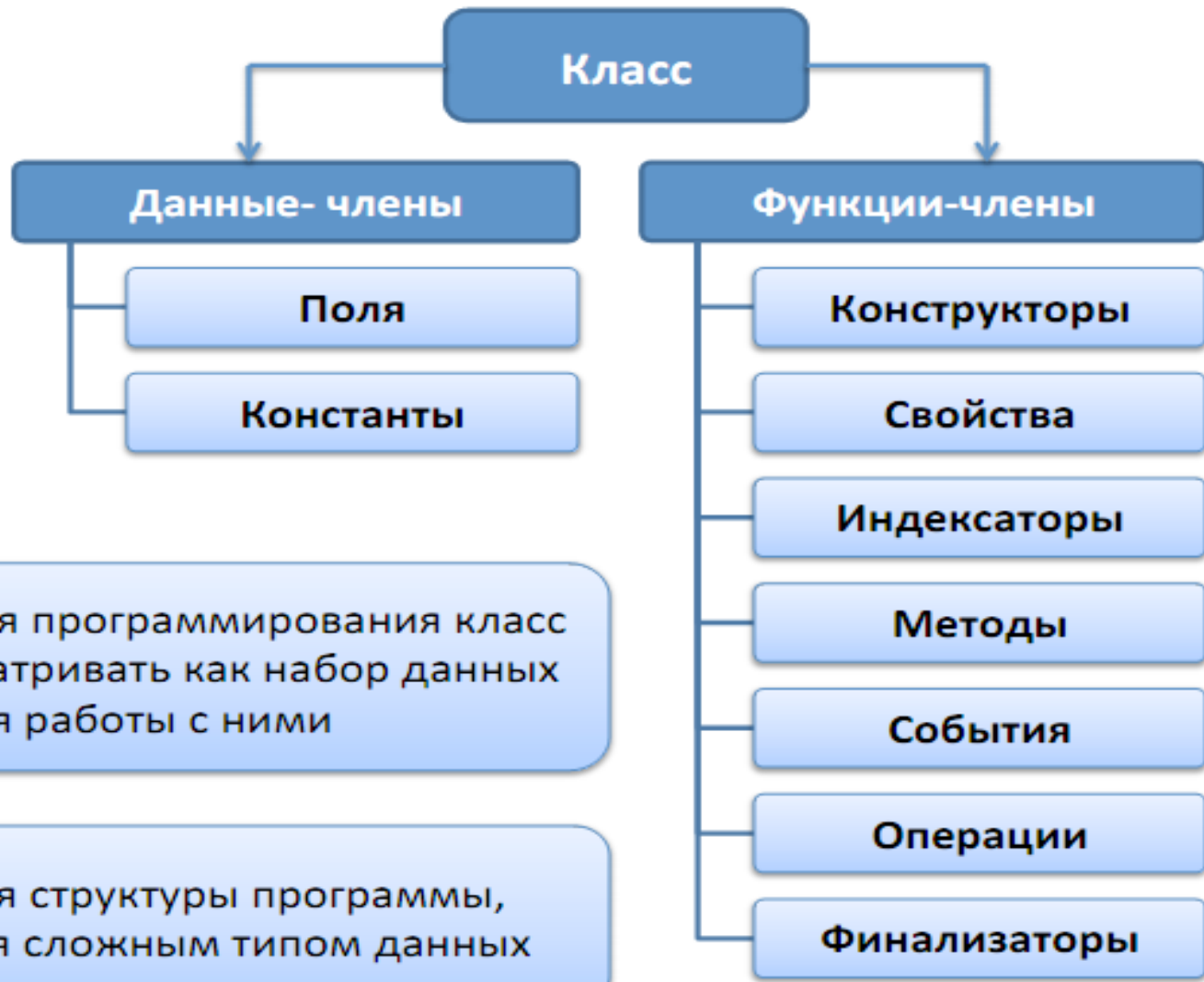


# NET.C#.04

## КЛАСС И СТРУКТУРЫ КОНСТРУКТОР И КОНСТРУКТОР ТИПА SINGLETON

**Author: Саркисян Гаяне Феликсовна**  
**[gayane.f.sarkisyan@gmail.com](mailto:gayane.f.sarkisyan@gmail.com)**

# Что такое тип?



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

С точки зрения структуры программы, класс является сложным типом данных

## Что такое класс?

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

```
class House
{
    ...
}
```

Класс определяется с ключевым словом **class**

oneHouse

twoHouse

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

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

С точки зрения структуры программы, класс является сложным типом данных

## Класс, объект, ссылка

**Объект** – это понятие времени выполнения, любой объект является экземпляром класса, создается во время выполнения системы и представляет набор полей

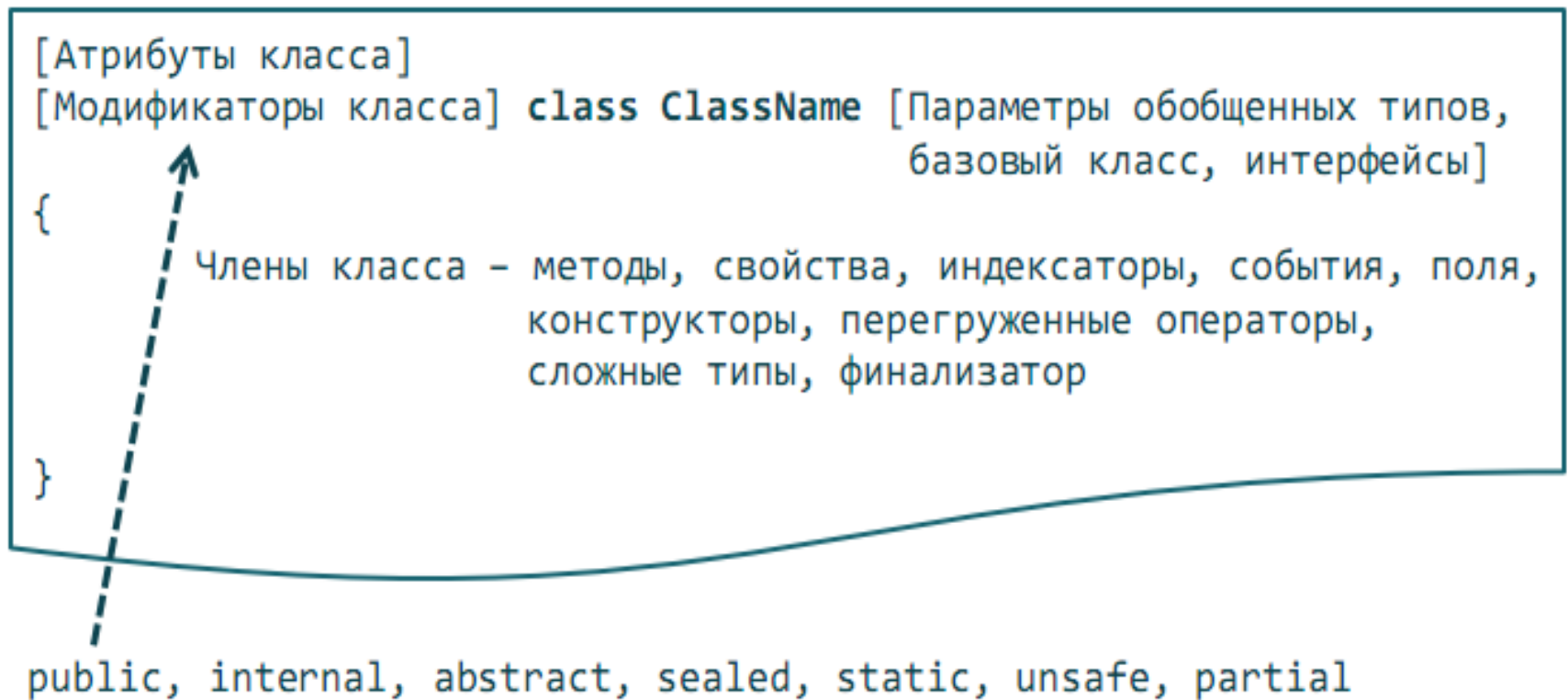
**Ссылка** - это понятие времени выполнения. Значение ссылки либо null, либо она присоединена к объекту, который она однозначно идентифицирует

**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

## Что такое класс?



## Члены класса

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

**Метод** это процедура или функция, определенная внутри класса

Статический модификатор	<code>static</code>
Модификатор доступа	<code>public internal private protected</code>
Модификатор наследования	<code>new virtual abstract override sealed</code>
Модификатор неуправляемого кода	<code>unsafe extern</code>

## Добавление элементов в классы

```
public class Residence  
{
```

```
    private ResidenceType type;  
    private int numberOfBedrooms;  
    private bool hasGarage;  
    private bool hasGarden;
```

Поля

```
    public int CalculateSalePrice()  
    {  
        // Code to calculate the sale value of the residence.  
    }
```

```
    public int CalculateRebuildingCost()  
    {  
        // Code to calculate the rebuilding costs of the  
        residence.  
    }  
}
```

Методы

```
public enum ResidenceType  
{  
    House,  
    Flat,  
    Bungalow,  
    Apartment  
};
```



## Определение конструкторов и инициализация объектов

Конструкторы — это специальные методы, *позволяющие корректно инициализировать объект класса* при его создании.

Конструкторы **не наследуются** и следовательно к ним **нельзя применить** модификаторы наследования такие как *virtual, new, override, sealed u abstract*

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

## Определение конструкторов и инициализация объектов

При определении конструктора соблюдаются следующие правила и принципы:

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

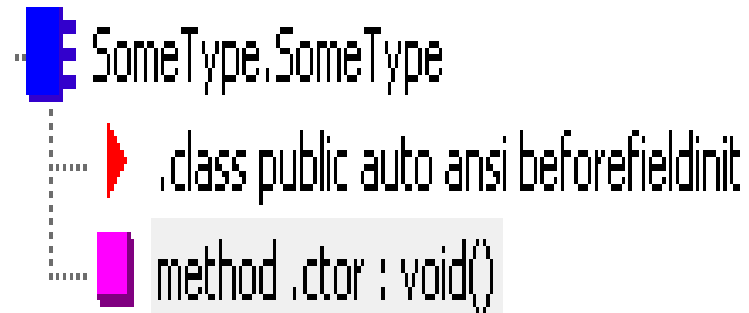
Конструкторы, как правило, объявляются с модификатором доступа `public`, чтобы любая часть приложения имела доступ к ним для создания и инициализации объектов

Конструкторы обычно инициализируют некоторые или все поля объекта, а также могут выполнять любые дополнительные задачи инициализации, требуемые классу

# Определение конструкторов и инициализация объектов

Конструктор по умолчанию автоматически генерирует для *ссылочного типа* тогда и только тогда, когда в нем не было определено **ни одного конструктора**.

```
namespace SomeType
{
    0 references
    public class SomeType
    {
    }
}
```



```
.method public hidebysig specialname rtspecialname
    instance void .ctor() cil managed
{
    // Code size          7 (0x7)
    .maxstack 8
    IL_0000: ldarg.0
    IL_0001: call         instance void [mscorlib]System.Object::.ctor()
    IL_0006: ret
} // end of method SomeType::.ctor
```

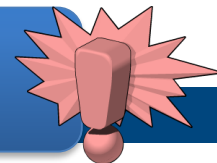
## Определение конструкторов и инициализация объектов

Для обеспечения того, чтобы объект был полностью инициализирован и все его поля имели значимые значения, в классе следует определить один или несколько конструкторов

```
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)
    {
    }
}
```

Конструкторы

При создании объекта CLR вызывает конструктор автоматически



## Определение конструкторов и инициализация объектов

```
public Residence(ResidenceType type, int numberOfBedrooms, bool hasGarden)
{
    this.type = type;
    this.numberOfBedrooms = numberOfBedrooms;
    this.hasGarden = hasGarden;
}
```

```
Residence House1 = new Residence(ResidenceType.House, 3)
    { hasGarden = true };
```

При создании объекта с помощью инициализатора объекта работает соответствующий конструктор, а затем полям присваиваются значения

## Определение конструкторов и инициализация объектов

Для использования переменной класса необходимо создать экземпляр соответствующего класса и присвоить его переменной класса

```
Residence myFlat = new Residence(ResidenceType.Flat, 2);  
Residence myHouse = new Residence(ResidenceType.House, 3, true);  
Residence myBungalow = new Residence(ResidenceType.Bungalow, 2, true, true);
```

```
Residence myFlat = new Residence( );
```


**СТЕ**

Объект может иметь большое количество полей, и не всегда возможно или целесообразно предусматривать конструкторы, которые могут инициализировать их все возможные комбинации

## Определение конструкторов и инициализация объектов

```
public class Residence
{
    private ResidenceType type;
    private int numberOfBedrooms;
    private bool hasGarage;
    private bool hasGarden;

    public 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)
    {
    }
    ...
}
```



Реализация конструктора по умолчанию, вызывающего параметризованный конструктор с множеством значений по умолчанию для каждого параметра

## Определение конструкторов и инициализация объектов

### Инициализация полей ссылочного типа.

```
public class SomeRefType
{
    private int x = 45;

    0 references
    public SomeRefType()
    {
    }

    0 references
    public SomeRefType(int a)
    {
        x = a;
    }
}
```



# Определение конструкторов и инициализация объектов

## Инициализация полей ссылочного типа.

```
public class SomeRefType
{
    private int x = 45;

    0 references
    public SomeRefType()
    {
    }

    0 references
    public SomeRefType(int a)
    {
        x = a;
    }
}
```

```
.method public hidebysig specialname rtspecialname
    instance void .ctor() cil managed
{
    // Code size          18 (0x12)
    .maxstack 8
    IL_0000: ldarg 0
    IL_0001: ldc.i4.s    45
    IL_0003: stfld     int32 SomeType.SomeRefType::x
    IL_0008: ldarg.0
    IL_0009: call        instance void [mscorlib]System.Object::.ctor()
    IL_000e: nop
    IL_000f: nop
    IL_0010: nop
    IL_0011: ret
} // end of method SomeRefType::.ctor
```

# Определение конструкторов и инициализация объектов

## Инициализация полей ссылочного типа.

```
public class SomeRefType
{
    private int x = 45;
```

0 references

```
public SomeRefType()
{
}
}
```

0 references

```
public SomeRefType(int a)
{
    x = a;
}
}
```

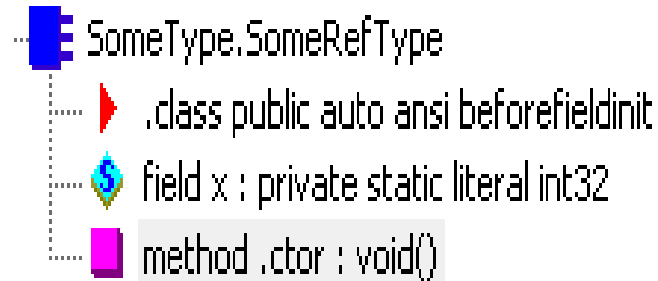
```
.method public hidebysig specialname rtspecialname
    instance void .ctor(int32 a) cil managed
{
    // Code size          25 (0x19)
    .maxstack 8
    IL_0000: ldarg.0
    IL_0001: ldc.i4.s    45
    IL_0003: stfld      int32 SomeType.SomeRefType::x
    IL_0008: ldarg.0
    IL_0009: call        instance void [mscorlib]System.Object::.ctor()
    IL_000e: nop
    IL_000f: nop
    IL_0010: ldarg.0
    IL_0011: ldarg.1
    IL_0012: stfld      int32 SomeType.SomeRefType::x
    IL_0017: nop
    IL_0018: ret
} // end of method SomeRefType::.ctor
```

## Определение конструкторов и инициализация объектов

```
public class SomeRefType
{
    private int m_x;
    private string m_s;
    private double m_d;
    private byte m_b;
    3 references
    public SomeRefType()
    {
        m_x = 5;
        m_s = "Hi there";
        m_d = 3.14159;
        m_b = 0xff;
    }
    0 references
    public SomeRefType(int x) : this()
    {
        m_x = x;
    }
    0 references
    public SomeRefType(string s) : this()
    {
        m_s = s;
    }
    0 references
    public SomeRefType(int x, string s): this()
    {
        m_x = x;
        m_s = s;
    }
}
```

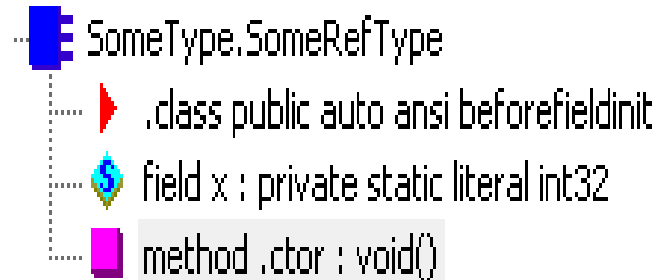
## Определение конструкторов и инициализация объектов

```
public class SomeRefType  
{  
    private const int x = 45;  
}
```



## Определение конструкторов и инициализация объектов

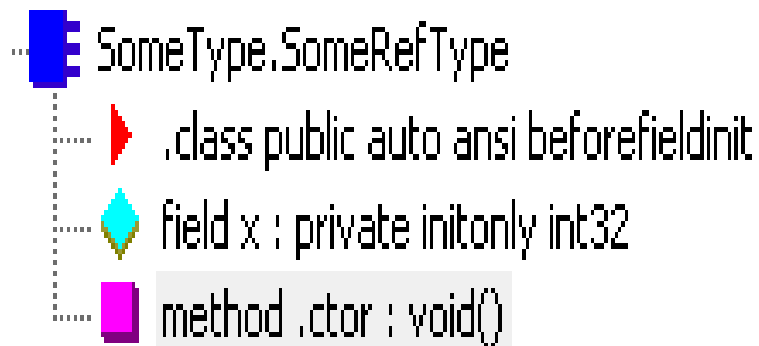
```
public class SomeRefType
{
    private const int x = 45;
}
```



```
.method public hidebysig specialname rtspecialname
    instance void .ctor() cil managed
{
    // Code size          7 (0x7)
    .maxstack 8
    IL_0000: ldarg.0
    IL_0001: call        instance void [mscorlib]System.Object::.ctor()
    IL_0006: ret
} // end of method SomeRefType::.ctor
```

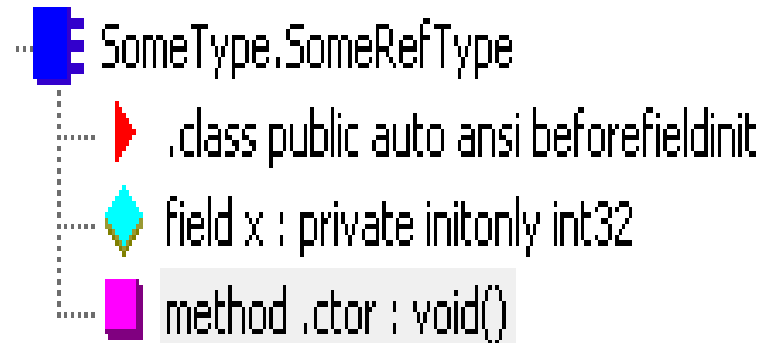
## Определение конструкторов и инициализация объектов

```
public class SomeRefType
{
    private readonly int x = 45;
}
```



## Определение конструкторов и инициализация объектов

```
public class SomeRefType
{
    private readonly int x = 45;
}
```



```
.method public hidebysig specialname rtspecialname
    instance void .ctor() cil managed
{
    // Code size          16 (0x10)
    .maxstack 8
    IL_0000: ldarg.0
    IL_0001: ldc.i4.s    45
    IL_0003: stfld      int32 SomeType.SomeRefType::x
    IL_0008: ldarg.0
    IL_0009: call       instance void [mscorlib]System.Object::.ctor()
    IL_000e: nop
    IL_000f: ret
} // end of method SomeRefType::.ctor
```

## Определение конструкторов и инициализация объектов

using System;

```
public class SomeRefType  
{  
    private static readonly int x = 45;  
}
```

- SomeType.SomeRefType
  - .class public auto ansi beforefieldinit
  - field x : private static initonly int32
  - method .cctor : void()
  - method .ctor : void()



# Определение конструкторов и инициализация объектов

using System;

```
public class SomeRefType
{
    private static readonly int x = 45;
}
```

```
.method private hidebysig specialname rtspecialname static
    void .cctor() cil managed
{
    // Code size      8 (0x8)
    .maxstack 8
    IL_0000: ldc.i4.s 45
    IL_0002: stsfld int32 SomeType.SomeRefType::x
    IL_0007: ret
} // end of method SomeRefType::cctor
```

■ SomeType.SomeRefType

- ▶ .class public auto ansi beforefieldinit
- ◆ field x : private static initonly int32
- ◆ method .cctor : void()
- ◆ method .ctor : void()

```
.method public hidebysig specialname rtspecialname
    instance void .ctor() cil managed
{
    // Code size      7 (0x7)
    .maxstack 8
    IL_0000: ldarg.0
    IL_0001: call instance void [mscorlib]System.Object::.ctor()
    IL_0006: ret
} // end of method SomeRefType::ctor
```

## Что такое структура?

Данные в переменных структурного типа хранятся своим значением

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

Структура может содержать поля и методы реализации

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



## Определение и использование структуры

Для объявления структуры используется ключевое слово **struct**

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

Синтаксис при определении членов в структурах аналогичен синтаксису в классах

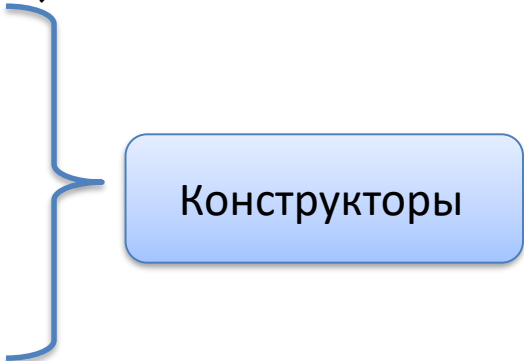
Для создания экземпляра типа структура необязательно использовать оператор `new`, однако структура в этом случае считается неинициализированной

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

## Инициализация структуры

Если при создании экземпляра необходимо инициализировать поля структуры, можно определить один или несколько конструкторов

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



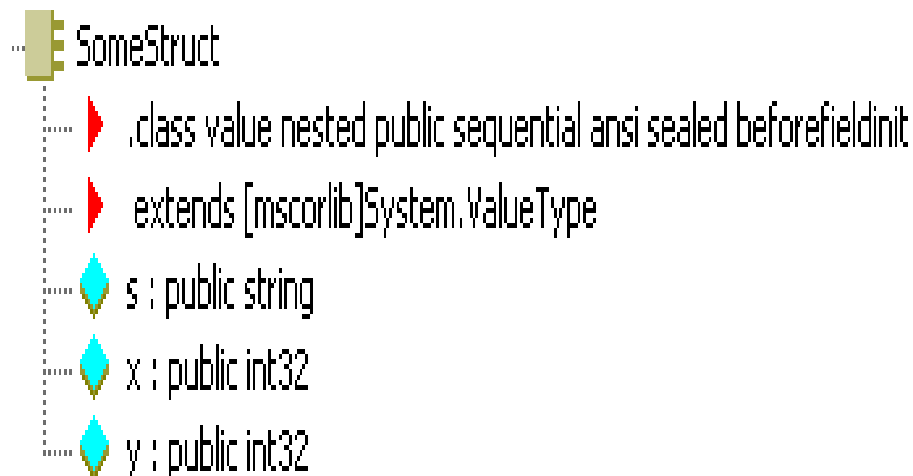
Конструкторы

*Сколько значимых типов из .NET Framework содержит конструкторы по умолчанию?*

## Определение конструкторов и инициализация значимых типов

```
public struct SomeStruct
{
    public int x, y;
    public string s;
}
```

```
static void Main(string[] args)
{
    SomeStruct p = new SomeStruct ();
}
```



```
.method private hidebysig static void Main(string[] args) cil managed
{
    .entrypoint
    // Code size          10 (0xa)
    .maxstack 1
    .locals init ([0] valuetype ConsoleApplication1.Program/SomeStruct p)
    IL_0000: nop
    IL_0001: ldloca.s    p
    IL_0003: initobj ConsoleApplication1.Program/SomeStruct
    IL_0009: ret
} // end of method Program::Main
```

# Определение конструкторов и инициализация значимых типов

```
public struct SomeStruct
{
    public int x, y;
    public string s;
    // В спецификации языка C# сказано, что пользователю запрещается
    // создавать конструктор по умолчанию явно, поскольку любая структура
    // содержит его неявно
}
static void Main()
{
    SomeStruct p = new SomeStruct (); //
    p.Dump("STRUCT    p=");

    SomeStruct p1 = default(SomeStruct);
    p1.Dump("STRUCT    p1=");
}
```

Results    λ    SQL    IL

**STRUCT p=**

SomeStruct	
UserQuery+SomeStruct	
x	0
y	0
s	null

**STRUCT p1=**

SomeStruct	
UserQuery+SomeStruct	
x	0
y	0
s	null


# Определение конструкторов и инициализация значимых типов

```
public struct SomeStruct
{
    public int x, y;
    public string s;
    // В спецификации языка C# сказано, что пользователю запрещается
    // создавать конструктор по умолчанию
    // содержит его неявно
}
static void Main()
{
    SomeStruct p = new SomeStruct (); //
    p.Dump("STRUCT p=");

    SomeStruct p1 = default(SomeStruct);
    p1.Dump("STRUCT p1=");
}
```

▼ Results λ SQL IL

IL_0000:	nop	
IL_0001:	ldloc.s	00 // p
IL_0003:	initobj	UserQuery.SomeStruct
IL_0009:	ldloc.0	// p
IL_000A:	ldstr	"STRUCT p="
IL_000F:	call	LINQPad.Extensions.Dump
IL_0014:	pop	
IL_0015:	ldloc.s	01 // p1
IL_0017:	initobj	UserQuery.SomeStruct
IL_001D:	ldloc.1	// p1
IL_001E:	ldstr	"STRUCT p1="
IL_0023:	call	LINQPad.Extensions.Dump
IL_0028:	pop	
IL_0029:	ret	



## Определение конструкторов и инициализация значимых типов

```
public struct Point
{
    int x = 1;

    int y;

    public Point() { }

    public Point (int x) { this.x = x; }
}
```



## Определение конструкторов и инициализация значимых типов

```
public struct SomeStruct
{
    public int x, y;
    public string s;

    public SomeStruct(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}

static void Main()
{
    SomeStruct p = new SomeStruct (1, 1);
    p.Dump("STRUCT    p=");
}
```

## Определение конструкторов и инициализация значимых типов

```
public struct SomeStruct
{
    public int x, y;
    public string s;

    public SomeStruct(int x, int y)
    {
        this.x = x;
        this.y = y;
        this.s = null;
    }
}

static void Main()
{
    SomeStruct p = new SomeStruct (1, 2);
    p.Dump("STRUCT    p=");
}
```

▼ Results λ SQL IL

```
IL_0000: nop
IL_0001: ldloc.s    00 // p
IL_0003: ldc.i4.1
IL_0004: ldc.i4.1
IL_0005: call        UserQuery+SomeStruct..ctor
IL_0017: ret

SomeStruct..ctor:
IL_0000: nop
IL_0001: ldarg.0
IL_0002: ldarg.1
IL_0003: stfld        UserQuery+SomeStruct.x
IL_0008: ldarg.0
IL_0009: ldarg.2
IL_000A: stfld        UserQuery+SomeStruct.y
IL_000F: ldarg.0
IL_0010: ldnull
IL_0011: stfld        UserQuery+SomeStruct.s
IL_0016: ret
```

## Определение конструкторов и инициализация значимых типов

```
public struct SomeStruct
{
    public int x, y;
    public string s;

    public SomeStruct(int x, int y) : this()
    {
        //this = new SomeStruct();
        this.x = x;
        this.y = y;
    }
}

static void Main()
{
    SomeStruct p = new SomeStruct (1, 1);
    p.Dump("STRUCT    p=");
}
```

## Определение конструкторов и инициализация значимых типов

```
public struct SomeStruct
{
    public int x, y;
    public string s;

    public SomeStruct(int x, int y) : this()
```

▼	Results	λ	SQL	IL
IL_0000:	nop			
IL_0001:	ldloc.s	00	// p	
IL_0003:	ldc.i4.1			
IL_0004:	ldc.i4.1			
IL_0005:	call		UserQuery+SomeStruct..ctor	
IL_0017:	ret			
SomeStruct..ctor:				
IL_0000:	ldarg.0			
IL_0001:	initobj		UserQuery.SomeStruct	
IL_0007:	nop			
IL_0008:	ldarg.0			
IL_0009:	ldarg.1			
IL_000A:	stfld		UserQuery+SomeStruct.x	
IL_000F:	ldarg.0			
IL_0010:	ldarg.2			
IL_0011:	stfld		UserQuery+SomeStruct.y	
IL_0016:	nop			
IL_0017:	ret			

## Определение конструкторов и инициализация значимых типов

```
public struct Point
{
    public int x, y;
    public string s;

    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
        this.s = string.Empty;
    }
}
```

```
void Main()
{
    Rectangle r = new Rectangle();
    Point p = new Point();
    p.Dump();
    r.Dump();
}
```

```
public class Rectangle
{
    public string s;
    public int x;
    public Point topLeft, bottomRigth;

    public Rectangle(){ topLeft = new Point(); }
}
```

## Определение конструкторов и инициализация значимых типов

```
void Main()
{
    Rectangle r = new Rectangle();
    Point p = new Point();
    p.Dump();
    r.Dump();
}
```

▼ Results λ SQL IL

Point ▶	
UserQuery+Point	
x	0
y	0
s	<i>null</i>

Rectangle ▶											
UserQuery+Rectangle											
s	<i>null</i>										
x	0										
topLeft	<table><tr><th colspan="2">Point ▶</th></tr><tr><th colspan="2">UserQuery+Point</th></tr><tr><td>x</td><td>0</td></tr><tr><td>y</td><td>0</td></tr><tr><td>s</td><td><i>null</i></td></tr></table>	Point ▶		UserQuery+Point		x	0	y	0	s	<i>null</i>
Point ▶											
UserQuery+Point											
x	0										
y	0										
s	<i>null</i>										
bottomRigth	<table><tr><th colspan="2">Point ▶</th></tr><tr><th colspan="2">UserQuery+Point</th></tr><tr><td>x</td><td>0</td></tr><tr><td>y</td><td>0</td></tr><tr><td>s</td><td><i>null</i></td></tr></table>	Point ▶		UserQuery+Point		x	0	y	0	s	<i>null</i>
Point ▶											
UserQuery+Point											
x	0										
y	0										
s	<i>null</i>										

## Определение конструкторов и инициализация значимых типов

```
void Main()
{
    Rectangle r = new Rectangle();
    Point p = new Point();
    p.Dump();
    r.Dump();
}
```

```
IL_0000: nop
IL_0001: newobj      UserQuery+Rectangle..ctor
IL_0006: stloc.0      // r
IL_0007: ldloc.s      01 // p
IL_0009: initobj      UserQuery.Point
IL_000F: ldloc.1      // p
IL_0010: call         LINQPad.Extensions.Dump
IL_0015: pop
IL_0016: ldloc.0      // r
IL_0017: call         LINQPad.Extensions.Dump
IL_001C: pop
IL_001D: ret
```

## Определение конструкторов и инициализация значимых типов

```
public class Rectangle
{
    public string s;
    public int x;
    public Point topLeft, bottomRigth;

    public Rectangle(){ topLeft = new Point(); }
}
```

Rectangle..ctor:

```
IL_0000: ldarg.0
IL_0001: call         System.Object..ctor
IL_0006: nop
IL_0007: nop
IL_0008: ldarg.0
IL_0009: ldflda         UserQuery+Rectangle.topLeft
IL_000E: initobj        UserQuery.Point
IL_0014: nop
IL_0015: ret
```



## Определение конструкторов и инициализация значимых типов

```
public struct Point
{
    public int x, y;
    public string s;

    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
        this.s = string.Empty;
    }
}
```

**Point..ctor:**

```
IL_0000:  nop
IL_0001:  ldarg.0
IL_0002:  ldarg.1
IL_0003:  stfld      UserQuery+Point.x
IL_0008:  ldarg.0
IL_0009:  ldarg.2
IL_000A:  stfld      UserQuery+Point.y
IL_000F:  ldarg.0
IL_0010:  ldsfld     System.String.Empty
IL_0015:  stfld      UserQuery+Point.s
IL_001A:  ret
```

## Определение конструкторов и инициализация значимых типов

```
public struct Point
{
    public int x, y;
    public string s;

    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
        this.s = string.Empty;
    }
}
```

```
void Main()
{
    Rectangle r = new Rectangle();
    Point p = new Point();
    p.Dump();
    r.Dump();
}
```

```
public class Rectangle
{
    public string s;
    public int x;
    public Point topLeft, bottomRigth;
}
```

## Определение конструкторов и инициализация значимых типов

```
void Main()
{
    Rectangle r = new Rectangle();
    Point p = new Point();
    p.Dump();
    r.Dump();
}
```

▼ Results λ SQL IL

Point ▶	
UserQuery+Point	
x	0
y	0
s	null

Rectangle ▶											
UserQuery+Rectangle											
s	null										
x	0										
topLeft	<table><tr><th colspan="2">Point ▶</th></tr><tr><th colspan="2">UserQuery+Point</th></tr><tr><td>x</td><td>0</td></tr><tr><td>y</td><td>0</td></tr><tr><td>s</td><td>null</td></tr></table>	Point ▶		UserQuery+Point		x	0	y	0	s	null
Point ▶											
UserQuery+Point											
x	0										
y	0										
s	null										
bottomRigth	<table><tr><th colspan="2">Point ▶</th></tr><tr><th colspan="2">UserQuery+Point</th></tr><tr><td>x</td><td>0</td></tr><tr><td>y</td><td>0</td></tr><tr><td>s</td><td>null</td></tr></table>	Point ▶		UserQuery+Point		x	0	y	0	s	null
Point ▶											
UserQuery+Point											
x	0										
y	0										
s	null										

## Определение конструкторов и инициализация значимых типов

```
public class Rectangle
{
    public string s;
    public int x;
    public Point topLeft, bottomRigth;
}
```

### Rectangle..ctor:

IL\_0000: ldarg.0

IL\_0001: call System.Object..ctor

IL\_0006: nop

IL\_0007: ret

## Определение конструкторов и инициализация значимых типов

```
struct Mutable
{
    public Mutable(int x, int y)
        : this()
    {
        X = x;
        Y = y;
    }
    public void IncrementX() { X++; }
    public int X { get; private set; }
    public int Y { get; set; }
}
```

## Определение конструкторов и инициализация значимых типов

```
struct Mutable
```

```
{
```

```
    public Mutable(int x, int y)
```

```
    {  
        : this()  
    {
```

```
        X = x;
```

```
        Y = y;
```

```
    }
```

```
    public void IncrementX() { X++; }
```

```
    public int X { get; private set; }
```

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

```
}
```

Mutable.get\_X:

IL\_0000: ldarg.0

IL\_0001: ldfld UserQuery+Mutable.<X>k\_\_BackingField

IL\_0006: ret

Mutable.set\_X:

IL\_0000: ldarg.0

IL\_0001: ldarg.1

IL\_0002: stfld UserQuery+Mutable.<X>k\_\_BackingField

IL\_0007: ret

## Определение конструкторов и инициализация значимых типов

```
struct Mutable
{
    public Mutable(int x, int y)
        : this()
    {
        X = x;
        Y = y;
    }
    public void IncrementX() { X++; }
    public int X { get; private set; }
    public int Y { get; set; }
}
```

```
Mutable.get_Y:
IL_0000: ldarg.0
IL_0001: ldfld      UserQuery+Mutable.<Y>k__BackingField
IL_0006: ret
```

```
Mutable.set_Y:
IL_0000: ldarg.0
IL_0001: ldarg.1
IL_0002: stfld      UserQuery+Mutable.<Y>k__BackingField
IL_0007: ret
```

## Определение конструкторов и инициализация значимых типов

```
struct Mutable
```

```
{
```

```
    public Mutable(int x, int y)
```

```
    {  
        : this()
```

```
    {
```

```
        X = x;
```

```
        Y = y;
```

```
    }
```

```
    public void IncrementX() { X++; }
```

```
    public int X { get; private set; }
```

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

```
}
```

Mutable..ctor:

IL\_0000: ldarg.0

IL\_0001: initobj      UserQuery.Mutable

IL\_0007: nop

IL\_0008: ldarg.0

IL\_0009: ldarg.1

IL\_000A: call      UserQuery+Mutable.set\_X

IL\_000F: nop

IL\_0010: ldarg.0

IL\_0011: ldarg.2

IL\_0012: call      UserQuery+Mutable.set\_Y

IL\_0017: nop

IL\_0018: ret



## Определение конструкторов и инициализация значимых типов

```
struct Mutable
```

```
{  
    public Mutable(int x, int y)  
        : this()  
    {  
        X = x;  
        Y = y;  
    }  
    public void IncrementX() { X++; }  
    public int X { get; private set; }  
    public int Y { get; set; }  
}
```

Mutable.IncrementX:

```
IL_0000: nop  
IL_0001: ldarg.0  
IL_0002: call        UserQuery+Mutable.get_X  
IL_0007: stloc.0  
IL_0008: ldarg.0  
IL_0009: ldloc.0  
IL_000A: ldc.i4.1  
IL_000B: add  
IL_000C: call        UserQuery+Mutable.set_X  
IL_0011: nop  
IL_0012: ret
```

## Определение конструкторов и инициализация значимых типов

```
void Main()
{
    Mutable ob = new Mutable(1, 1);
    ob.Y++;
    ob.X++;
    Console.WriteLine("X=" + ob.X);
    Console.WriteLine("Y=" + ob.Y);
}
```

## Определение конструкторов и инициализация значимых типов

```
void Main()
{
    Mutable ob = new Mutable(1, 1);
    ob.Y++;
    ob.IncrementX();
    Console.WriteLine("X=" + ob.X);
    Console.WriteLine("Y=" + ob.Y);
}
```

## Определение конструкторов и инициализация значимых типов

```
class A
{
    public A() { Mutable = new Mutable(x: 5, y: 5); }
    public Mutable Mutable { get; set; }
}
```

```
A..ctor:
IL_0000: ldarg.0
IL_0001: call        System.Object..ctor
IL_0006: nop
IL_0007: nop
IL_0008: ldarg.0
IL_0009: ldc.i4.5
IL_000A: ldc.i4.5
IL_000B: newobj      UserQuery+Mutable..ctor
IL_0010: call        UserQuery+A.set_Mutable
IL_0015: nop
IL_0016: ret
```

## Определение конструкторов и инициализация значимых типов

```
class A
{
    public A() { Mutable = new Mutable(x: 5, y: 5); }
    public Mutable Mutable { get; set; }
}
```

A.get\_Mutable:

IL\_0000: ldarg.0

IL\_0001: ldfld UserQuery+A.<Mutable>k\_\_BackingField

IL\_0006: ret

A.set\_Mutable:

IL\_0000: ldarg.0

IL\_0001: ldarg.1

IL\_0002: stfld UserQuery+A.<Mutable>k\_\_BackingField

IL\_0007: ret

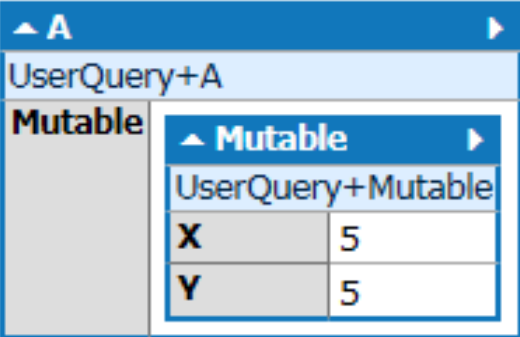
## Определение конструкторов и инициализация значимых типов

```
class A
{
    public A() { Mutable = new Mutable(x: 5, y: 5); }
    public Mutable Mutable { get; set; }
}

void Main()
{
    A a = new A();
    a.Mutable.Y.Dump();
    a.Dump();
}
```

IL\_0000: nop  
IL\_0001: newobj UserQuery+A..ctor  
IL\_0006: stloc.0 // a  
IL\_0007: ldloc.0 // a  
IL\_0008: callvirt UserQuery+A.get\_Mutable  
IL\_000D: stloc.1  
IL\_000E: ldloc.s 01  
IL\_0010: call UserQuery+Mutable.get\_Y  
IL\_0015: call LINQPad.Extensions.Dump<Int32>  
IL\_001A: pop  
IL\_001B: ldloc.0 // a  
IL\_001C: call LINQPad.Extensions.Dump<A>  
IL\_0021: pop  
IL\_0022: ret

5



A									
UserQuery+A									
Mutable	<table border="1"><thead><tr><th colspan="2">Mutable</th></tr><tr><th colspan="2">UserQuery+Mutable</th></tr></thead><tbody><tr><td>X</td><td>5</td></tr><tr><td>Y</td><td>5</td></tr></tbody></table>	Mutable		UserQuery+Mutable		X	5	Y	5
Mutable									
UserQuery+Mutable									
X	5								
Y	5								

## Определение конструкторов и инициализация значимых типов

```
class A
{
    public A() { Mutable = new Mutable(x: 5, y: 5); }
    public Mutable Mutable { get; set; }
}

void Main()
{
    A a = new A();
    a.Mutable.Y++;
    a.Mutable.IncrementX();
    a.Dump();
}
```

CTE

## Определение конструкторов и инициализация значимых типов

```
class A
{
    public A() { Mutable = new Mutable(x: 5, y: 5); }
    public Mutable Mutable { get; set; }
}


void Main()
{
    A a = new A();
    //a.Mutable.Y++;
    a.Mutable.IncrementX();
    a.Dump();
}
```

A	
UserQuery+A	
Mutable	Mutable
UserQuery+Mutable	
X	5
Y	5



## Определение конструкторов и инициализация значимых типов

```
| void Main()  
| {  
|     List<Mutable> lm = new List<Mutable> {  
|         new Mutable(x: 5, y: 5),  
|         new Mutable(x: 10, y: 10),  
|     };  
|  
|     lm.Dump();  
|     lm[1].Y++;  
|  
|     lm[1].IncrementX();  
|     lm.Dump();  
- }
```



## Определение конструкторов и инициализация значимых типов

```
void Main()
{
    Mutable[] lm = new Mutable []{
        new Mutable(x: 5, y: 5),
        new Mutable(x: 10, y: 20),
    };

    lm.Dump();
    lm[1].Y++;

    lm[1].IncrementX();
    lm.Dump();
}
```

▲ Mutable[] (2 items) ▶	
X ≡	Y ≡
5	5
10	20
15	25

▲ Mutable[] (2 items) ▶	
X ≡	Y ≡
5	5
11	21
16	26

## Определение конструкторов и инициализация значимых типов

```
void Main()
{
    Mutable[] lm = new Mutable []{new Mutable(x: 5, y: 5)};
    IList<Mutable> lst = lm;
    lst.Dump();

    //lst[0].Y++;

    lst[0].IncrementX();
    lst.Dump();
}
```

▲ Mutable[] (1 item) ▶	
X	Y
5	5

▲ Mutable[] (1 item) ▶	
X	Y
5	5

## Инициализация структуры

Существуют следующие различия между конструкторами структур и классов:

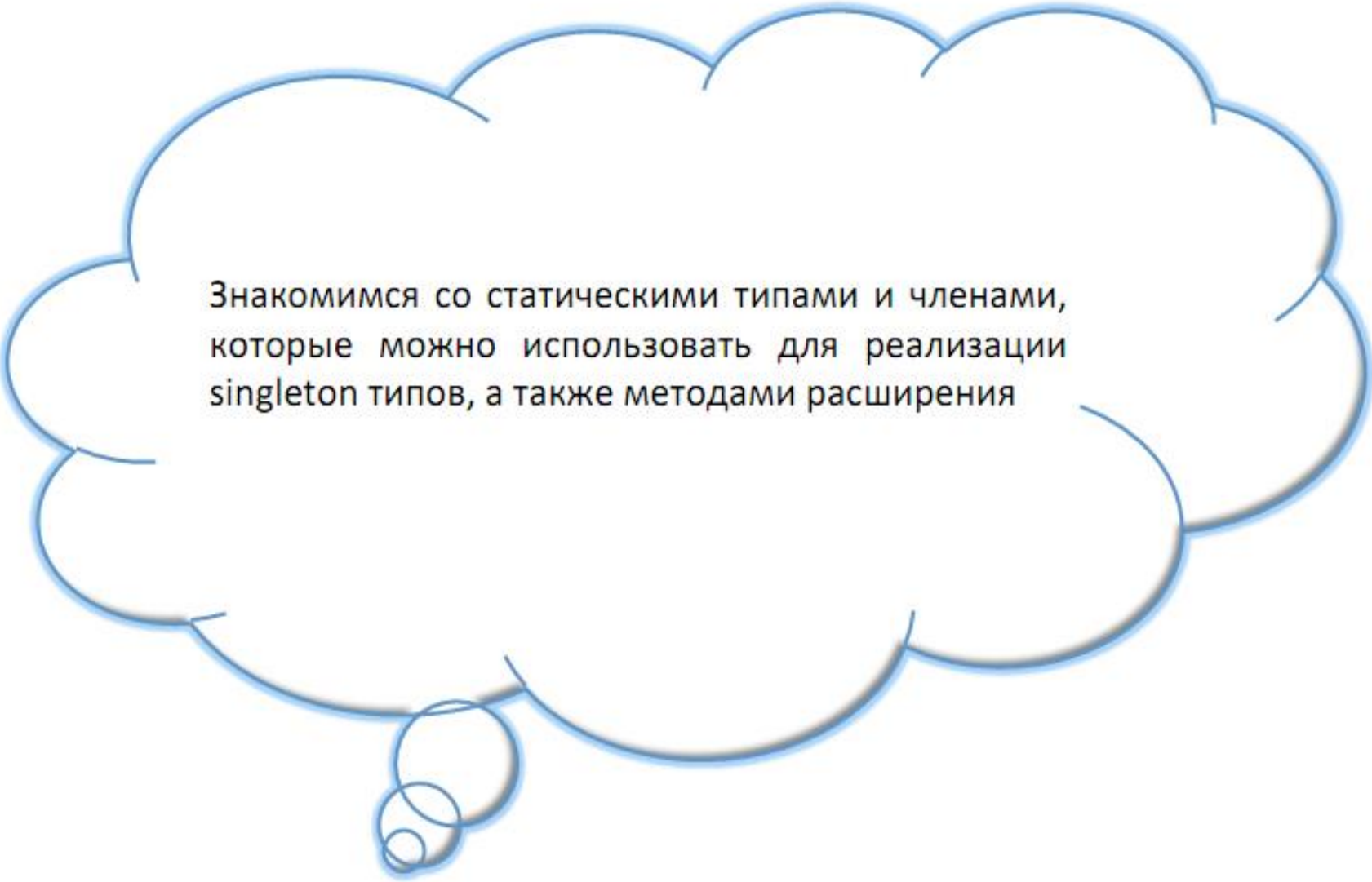
Для структуры нельзя определить конструктор по умолчанию

Все конструкторы структуры должны явно инициализировать каждое поле в структуре

Конструктор в структуре не может вызывать другие методы до присваивания значений всем ее полям

Если при создании экземпляра структуры не используется конструктор (либо default), структура считается неинициализированной





Знакомимся со статическими типами и членами, которые можно использовать для реализации singleton типов, а также методами расширения

## Создание и использование статических полей

Члены экземплярных типов обычно содержат данные и осуществляют функции, относящиеся к конкретному экземпляру типа

```
class Sales
{
    private double monthlyProfit;
    public void SetMonthlyProfit(double monthlyProfit)
    {
        this.monthlyProfit = monthlyProfit;
    }
    public double GetAnnualProfitForecast()
    {
        return (this.monthlyProfit * 12);
    }
}
```


```
Sales sales2010 = new Sales();
sales2010.SetMonthlyProfit(34672);
Console.WriteLine(sales2010.GetAnnualProfitForecast());
Sales sales2011 = new Sales();
sales2011.SetMonthlyProfit(98675);
Console.WriteLine(sales2011.GetAnnualProfitForecast());
```

## Создание и использование статических полей

Статические поля не принадлежат экземплярам типа, они принадлежат самому типу

Для создания статического поля при его объявлении необходимо использовать модификатор **static**

```
class Sales
{
    public static double salesTaxPercentage = 20;
}
```



```
Sales.salesTaxPercentage = 32;
```

Статический член создается, а память для него выделяется, когда в первый раз на него осуществляется ссылка

Нельзя получить доступ к статическим полям через экземпляр типа

```
Sales sales = new Sales();
sales.salesTaxPercentage = 23;
```

**CTE**



## Создание и использование статических полей

Получить доступ к статическим полям можно с помощью методов экземпляра и конструкторов, позволяющим совместно использовать данные несколькими экземплярами одного и того же типа

```
class User
{
    internal int usersOnline;
    internal User()
    {
        usersOnline++;
    }
}
User a = new User();
User b = new User();
User c = new User();
User d = new User();
int totalUsersOnline = d.usersOnline
```

```
class User
{
    internal static int usersOnline;
    internal User()
    {
        usersOnline++;
    }
}
User a = new User();
User b = new User();
User c = new User();
User d = new User();
int totalUsersOnline = User.usersOnline;
```

totalUsersOnline =






## Создание и использование статических методов

Статические методы как правило используются в классах для выполнения атомарных операций, не полагающихся на данные экземпляра

Чтобы определить статический метод в объявлении метода следует использовать ключевое слово `static`

```
class Sales
{
    public static double GetMonthlySalesTax(double monthlyProfit)
    {...}
}
```



```
class Sales
{
    private static double salesTaxPercentage = 20;
    public static double GetMonthlySalesTax(double monthlyProfit)
    {
        return (salesTaxPercentage * monthlyProfit) / 100;
    }
}
```

Статические методы могут использовать только данные, хранящиеся в статических полях и данные, которые передаются в качестве параметров в сигнатуре метода

## Создание статических типов и использование статических конструкторов

Тип может содержать как статические члены, так и члены экземпляра

При разработке служебного класса, содержащего только статические члены, можно объявить сам класс как статический (не к структурным типам!)

Для объявления статического типа при его объявлении используется модификатор **static**

Типы могут содержать статические конструкторы (конструкторы классов или инициализаторы типов)

```
static class(!) Sales
{
    ...
}
```

```
class Sales
{
    static Sales()
    {
        ...
    }
}
```

- ✓ Создание экземпляра невозможно
- ✓ Не наследует интерфейсов
- ✓ В нем можно использовать только статические члены
- ✓ Не может быть членом поля, параметром метода или локальной переменной

CLR неявно вызывает статический конструктор перед тем, как любой код пытается получить доступ к статическому члену в этом типе

## Создание статических типов и использование статических конструкторов

При использовании статических конструкторов типов необходимо следовать некоторым правилам, чтобы избежать ошибок компиляции

Можно определить только один конструктор, имеющий префикс-модификатор `static`

Статический конструктор всегда использует неявный модификатор доступа `private`

Нельзя указать список параметров

Статический конструктор может содержать только ссылки на другие статические члены

## Создание статических типов и использование статических конструкторов

Статический конструктор для класса выполняется не более одного раза в домене приложения. Выполнение статического конструктора инициируется одним из следующих событий:

- создается экземпляр класса
- происходит обращение к любому статическому члену класса

1. A type may have a type-initializer method, or not.
2. A type may be specified as having a relaxed semantic for its type-initializer method (for convenience below, we call this relaxed semantic `BeforeFieldInit`)
3. If marked `BeforeFieldInit` then the type's initializer method is executed at, or sometime before, first access to any static field defined for that type
4. If not marked `BeforeFieldInit` then that type's initializer method is executed at (i.e., is triggered by):
  - first access to any static or instance field of that type, or
  - first invocation of any static, instance or virtual method of that type

CLI specification (ECMA 335) states in section 8.9.5

### *LINQPadQueries.StaticCtors*

#### *1\_1\_Ref*

```
public class SomeRefType
{
    public static int xStatic = 10;
    public int xInstance = 45;

    public SomeRefType()
    {
        Console.WriteLine("Instance ctor for class works!");
    }
}

void Main()
{
    SomeRefType b = new SomeRefType();
    SomeRefType c = new SomeRefType();
}
```

## Создание статических типов и использование статических конструкторов

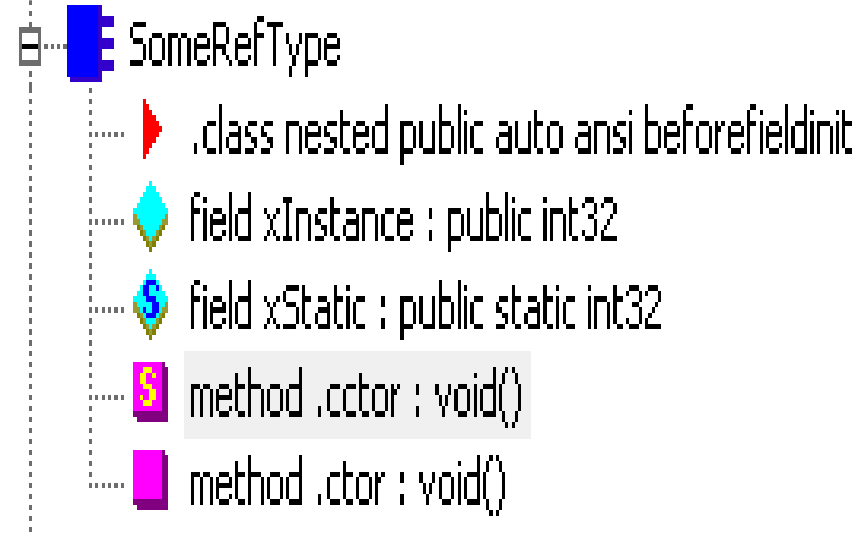
### *LINQPadQueries.StaticCtors*

1\_1\_Ref

```
public class SomeRefType
{
    public static int xStatic = 10;
    public int xInstance = 45;

    public SomeRefType()
    {
        Console.WriteLine("Ins
    }
}

void Main()
{
    SomeRefType b = new SomeRefType();
    SomeRefType c = new SomeRefType();
}
```



The image shows a LINQPad query editor with a C# class `SomeRefType` and its `Main` method. The class has two public integer fields, `xStatic` and `xInstance`, and a public constructor. The `Main` method creates two instances of `SomeRefType`. To the right of the code, a tree view displays the structure of the `SomeRefType` class. The tree view includes a root node for the class, followed by a list of members: a red triangle icon for the class declaration, a blue diamond icon for the field `xInstance`, a blue diamond icon for the field `xStatic`, a yellow square icon for the constructor `.ctor`, and a yellow square icon for the constructor `.ctor`.

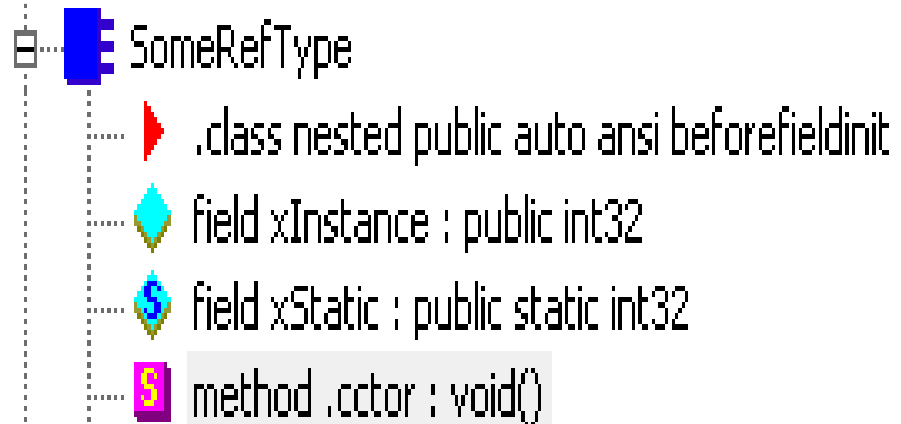
## Создание статических типов и использование статических конструкторов

### LINQPadQueries.StaticCtors

#### 1\_1\_Ref

```
public class SomeRefType
{
    public static int xStatic = 10;
    public int xInstance = 45;

    public SomeRefType()
    {
        Console.WriteLine("Instance ct
```



SomeRefType::method .cctor : void()

Find Find Next

```
.method private hidebysig specialname rtspecialname static
    void .cctor() cil managed
{
    // Code size      8 (0x8)
    .maxstack 8
    IL_0000: ldc.i4.s    10
    IL_0002: stsfld      int32 ConsoleApplication1.Program/SomeRefType::xStatic
    IL_0007: ret
} // end of method SomeRefType::.cctor
```

## Создание статических типов и использование статических конструкторов

### *LINQPadQueries.StaticCtors*

#### *1\_2\_Ref*

```
public class SomeRefType
{
    public static int xStatic;
    public int xInstance = 45;

    public SomeRefType()
    {
        Console.WriteLine("Instance ctor
                           for class works!");
    }
}

void Main()
{
    SomeRefType b = new SomeRefType();
    (SomeRefType.xStatic).Dump();
    SomeRefType c = new SomeRefType();
}
```



## Создание статических типов и использование статических конструкторов

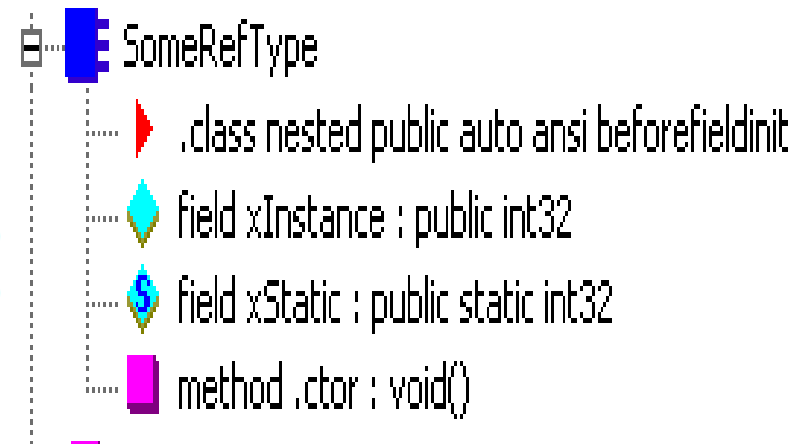
### *LINQPadQueries.StaticCtors*

1\_2\_Ref

```
public class SomeRefType
{
    public static int xStatic;
    public int xInstance = 45;

    public SomeRefType()
    {
        Console.WriteLine("In:
                           for class worl
    }
}

void Main()
{
    SomeRefType b = new SomeRefType();
    (SomeRefType.xStatic).Dump();
    SomeRefType c = new SomeRefType();
}
```



Results λ SQL IL

```
Instance ctor for class works!
0
Instance ctor for class works!
```

## Создание статических типов и использование статических конструкторов

### *LINQPadQueries.StaticCtors* 1\_3\_Ref

```
public class SomeRefType
{
    public static int xStatic = 67;
    public int xInstance = 45;

    public SomeRefType()
    {
        Console.WriteLine("Instance ctor
                           for class works!");
    }

    static SomeRefType()
    {
        Console.WriteLine("Static ctor
                           for class works!");
    }

    void Main()
    {
        SomeRefType b = new SomeRefType();
        SomeRefType c = new SomeRefType();
    }
}
```

## Создание статических типов и использование статических конструкторов

### *LINQPadQueries.StaticCtors* 1\_3\_2016

```
public class SomeRefType
{
    public static int xStatic = 1;
    public int xInstance = 45;

    public SomeRefType()
    {
        Console.WriteLine("Instance ctor
                           for class works!");
    }

    static SomeRefType()
    {
        Console.WriteLine("Static ctor
                           for class works!");
    }
}

void Main()
{
    SomeRefType b = new SomeRefType();
    SomeRefType c = new SomeRefType();
}
```

Results

λ

SQL

IL

Static ctor for class works!  
Instance ctor for class works!  
Instance ctor for class works!

## Создание статических типов и использование статических конструкторов

### *LINQPadQueries.StaticCtors* 1\_3\_Ref

```
public class SomeRefType
{
    public static int xStatic = 67;
    public int xInstance = 45;

    public SomeRefType()
    {
        Console.WriteLine("Instance
                           for class works!");
    }
}
```

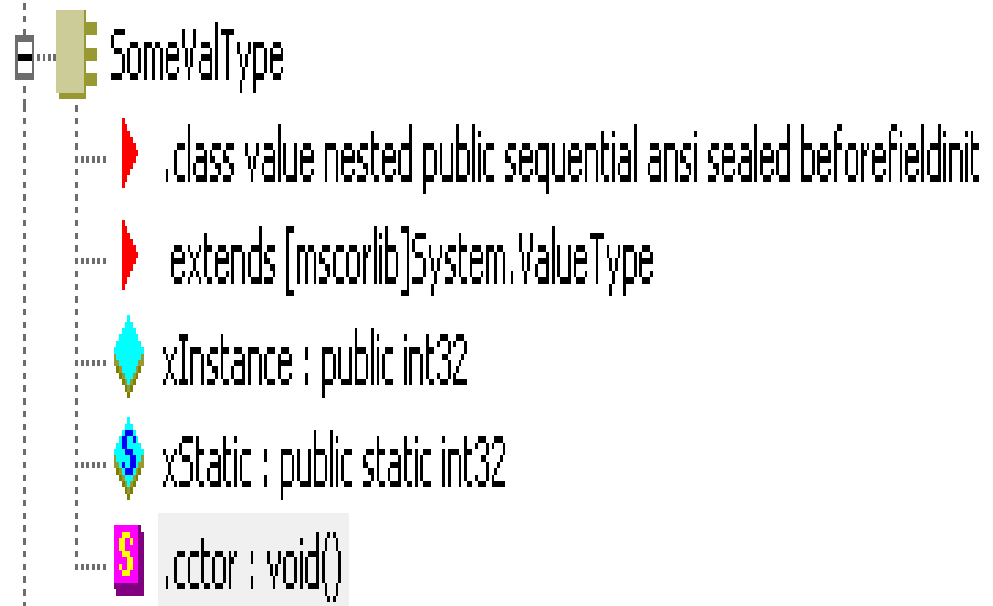


```
.method private hidebysig specialname rtspecialname static
    void .cctor() cil managed
{
    // Code size          21 (0x15)
    .maxstack 8
    IL_0000: ldc.i4.s      67
    IL_0002: stsfld          int32 ConsoleApplication1.Program/SomeRefType::xStatic
    IL_0007: nop
    IL_0008: ldstr            "Static ctor for class works!"
    IL_000d: call             void [mscorlib]System.Console::WriteLine(string)
    IL_0012: nop
    IL_0013: nop
    IL_0014: ret
} // end of method SomeRefType::.cctor
```

### *LINQPadQueries.StaticCtors* 1\_4\_Val

```
public struct SomeValType
{
    public static int xStatic = 123;
    public int xInstance;
}

void Main()
{
    SomeValType a = new SomeValType();
}
```



## Создание статических типов и использование статических конструкторов

### *LINQPadQueries.StaticCtors*

1\_4\_Val

```
public struct SomeValType
{
    public static int xStatic = 123;
    public int xInstance;
}
```

```
void Main()
{
```

```
.method private hidebysig specialname rtspecialname static
    void .cctor() cil managed
```

```
{
```

```
    // Code size          8 (0x8)
```

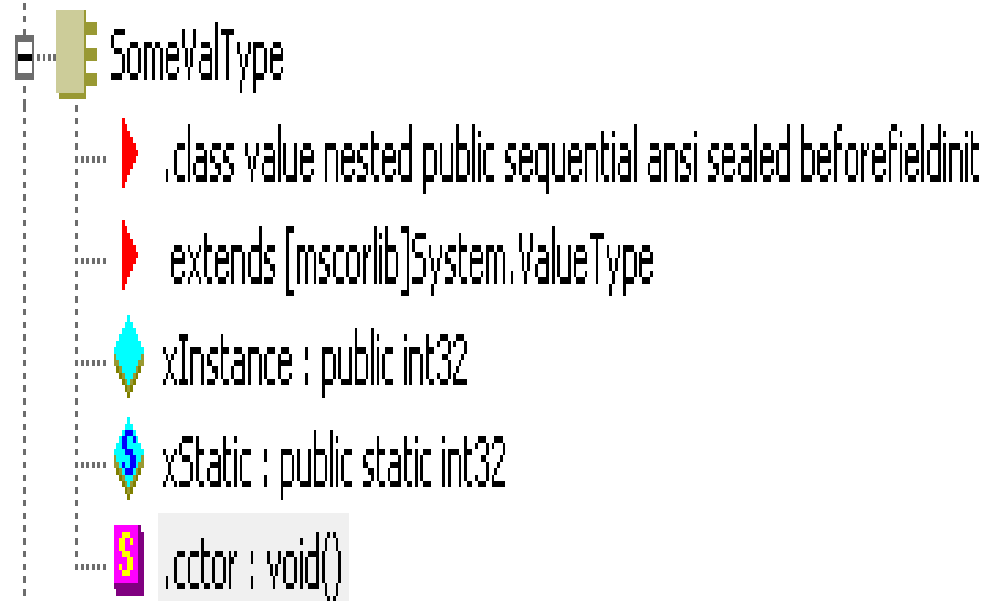
```
    .maxstack 8
```

```
IL_0000: ldc.i4.s    123
```

```
IL_0002: stsfld     int32 ConsoleApplication1.Program/SomeValType::xStatic
```

```
IL_0007: ret
```

```
} // end of method SomeValType::.cctor
```



## Создание статических типов и использование статических конструкторов

*LINQPadQueries.StaticCtors 1\_5\_Val*

```
public struct SomeValType
{
    public static int xStatic = 123;
    public int xInstance;
    static SomeValType(){
        Console.WriteLine("Static ctor for struct works!");
    }
}

void Main()
{
    SomeValType a = new SomeValType();
    SomeValType b;
    (a.xInstance = 7).Dump();
}
```

## Создание статических типов и использование статических конструкторов

### *LINQPadQueries.StaticCtors 1\_5\_Val*

```
public struct SomeValType
{
    public static int xStatic = 123;
    public int xInstance;
    static SomeValType(){
        Console.WriteLine("Static ctor for struct works!");
    }
}

void Main()
{
    SomeValType a = new SomeValType();
    SomeValType b;
    (a.xInstance = 7).Dump();
}
```

▼ Results λ SQL IL Tree

7



## Создание статических типов и использование статических конструкторов

*LINQPadQueries.StaticCtors 1\_6\_Val*

```
public struct SomeValType
{
    public static int xStatic = 123;
    public int xInstance;
    static SomeValType(){
        Console.WriteLine("Static ctor for struct works!");
    }
}

void Main()
{
    SomeValType a = new SomeValType();
    Console.WriteLine(SomeValType.xStatic);
    SomeValType b;
    SomeValType.xStatic = -90;
    Console.WriteLine(SomeValType.xStatic);
}
```

## Создание статических типов и использование статических конструкторов

*LINQPadQueries.StaticCtors 1\_6\_Val*

```
public struct SomeValType
{
    public static int xStatic = 123;
    public int xInstance;
    static SomeValType(){
        Console.WriteLine("Static ctor for struct works!");
    }
}

void Main()
{
    SomeValType a = new SomeValType();
    Console.WriteLine(SomeValType.xStatic);
    SomeValType b;
    SomeValType.xStatic = -90;
    Console.WriteLine(SomeValType.xStatic);
}
```

▼ Results λ SQL IL Tree

Static ctor for struct works!  
123  
-90

## Создание статических типов и использование статических конструкторов

### *LingPad 2.*

```
class Test
|{
|    static Test()
|    {
|        Console.WriteLine ("Type Initialized");
|    }
|}

static void Main()
|{
|    new Test();
|    new Test();
|    new Test();
|}
```

## Создание статических типов и использование статических конструкторов

### *LingPad 2.*

```
class Test
{
    static Test()
    {
        Console.WriteLine ("Type Initialized");
    }
}

static void Main()
{
    new Test();
    new Test();
    new Test();
}
```

---

Results   λ   SQL   IL

---

Type Initialized

## Создание статических типов и использование статических конструкторов

*LingPad 3.*

```
class Foo
{
    public static Foo Instance = new Foo();
    public static int x = 3;
    Foo() { Console.WriteLine (x); }
}

void Main()
{
    Console.WriteLine(Foo.x);
}
```

## Создание статических типов и использование статических конструкторов

```
class Foo
{
    public static Foo Instance = new Foo();
    public static int x = 3;
    Foo() { Console.WriteLine (x); }
}

void Main()
{
    Console.WriteLine(Foo.x);
}
```

---

Results	λ	SQL	IL
---------	---	-----	----

---

0  
3

## Создание статических типов и использование статических конструкторов

Foo

.class nested private auto ansi beforefieldinit

Foo::method .cctor : void()

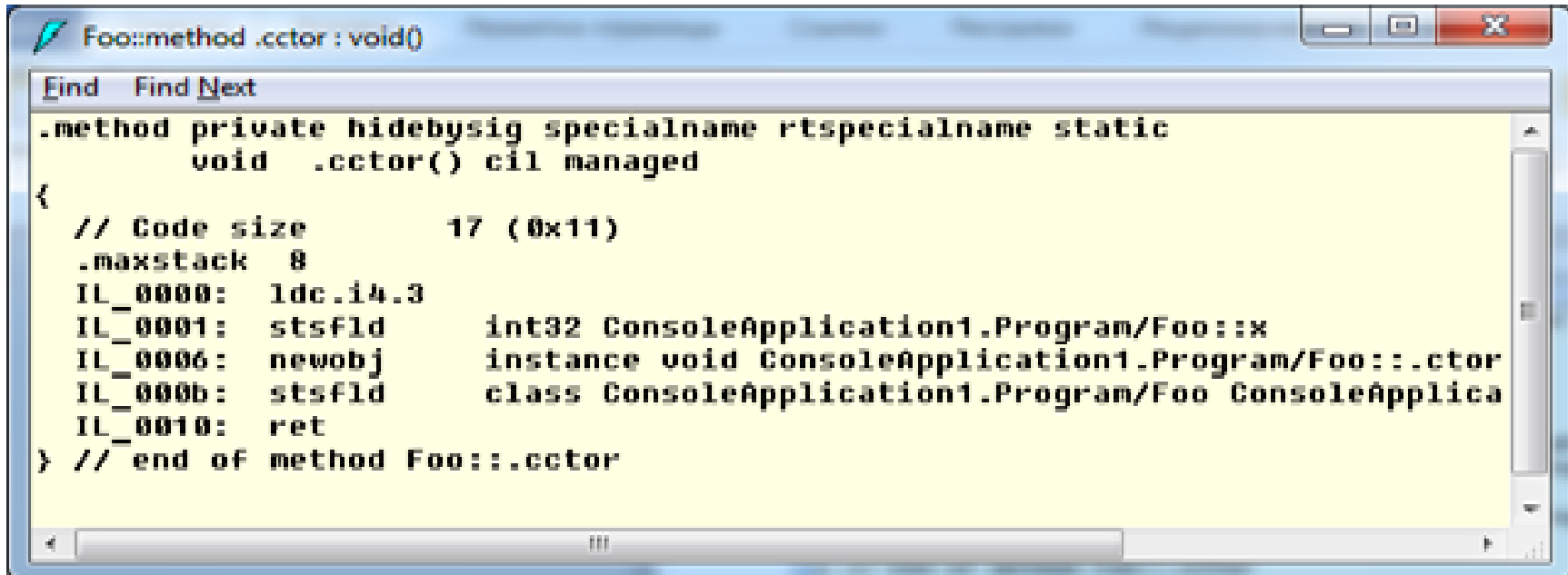
Find Find Next

```
.method private hidebysig specialname rtspecialname static
    void .cctor() cil managed
{
    // Code size      17 (0x11)
    .maxstack 8
    IL_0000: newobj      instance void ConsoleApplication1.Program/Foo::.cctor()
    IL_0005: stsfld      class ConsoleApplication1.Program/Foo ConsoleApplication1.Program/Foo::Instance
    IL_000a: ldc.i4.3
    IL_000b: stsfld      int32 ConsoleApplication1.Program/Foo::x
    IL_0010: ret
} // end of method Foo::.cctor
```

## Создание статических типов и использование статических конструкторов

```
class Foo
{
    public static int x = 3;
    public static Foo Instance = new Foo();
    Foo() { Console.WriteLine (x); }
}

void Main()
{
    Console.WriteLine(Foo.x);
}
```



The screenshot shows a debugger window titled "Foo::method .cctor : void()". The window contains a search bar with "Find" and "Find Next" buttons. Below the search bar, the IL code for the static constructor is displayed. The code is as follows:

```
.method private hidebysig specialname rtspecialname static
    void .cctor() cil managed
{
    // Code size          17 (0x11)
    .maxstack 8
    IL_0000: ldc.i4.3
    IL_0001: stsfld      int32 ConsoleApplication1.Program/Foo::x
    IL_0006: newobj       instance void ConsoleApplication1.Program/Foo::.ctor
    IL_000b: stsfld      class ConsoleApplication1.Program/Foo ConsoleApplica
    IL_0010: ret
} // end of method Foo::.cctor
```



## Создание статических типов и использование статических конструкторов

```
class Foo
{
    public static int x = 3;
    public static Foo Instance = new Foo();
    Foo() { Console.WriteLine (x); }
}

void Main()
{
    Console.WriteLine(Foo.x);
}
```

Results

λ

SQL

IL

3

3

## Создание статических типов и использование статических конструкторов

*LingPad 4.*

---

```
public class Test
{
    public static object _obj = new object();
}
static void Main()
{
    Test t = new Test();
}
```

## Создание статических типов и использование статических конструкторов

*LingPad 4.*

```
public class Test
{
    public static object _obj = new object();
}
```

Test::method .cctor: void()

Find Find Next

```
.method private hidebysig specialname rtspecialname static
    void .cctor() cil managed
{
    // Code size          11 (0xb)
    .maxstack 8
    IL_0000: newobj        instance void [mscorlib]System.Object::.ctor()
    IL_0005: stsfld          object ConsoleApplication1.Program/Test::_obj
    IL_000a: ret
} // end of method Test::.cctor
```

## Создание статических типов и использование статических конструкторов

*LingPad 5.*

```
class Test
{
    public static object _obj;

    static Test()
    {
        _obj = new object();
    }
}

static void Main()
{
    Test t = new Test();
}
```

## Создание статических типов и использование статических конструкторов

*LingPad 5.*

```
class Test
{
    public static object _obj;

    static Test()
    {
        _obj = new object();
    }
}
```

Test::method .cctor : void()

Find Find Next

```
.method private hidebysig specialname rtspecialname static
    void .cctor() cil managed
{
    // Code size          12 (0xc)
    .maxstack 8
    IL_0000: nop
    IL_0001: newobj      instance void [mscorlib]System.Object::.ctor()
    IL_0006: stsfld      object ConsoleApplication1.Program/Test::_obj
    IL_000b: ret
} // end of method Test::.cctor
```

## Создание статических типов и использование статических конструкторов

*LingPad 6.*

---

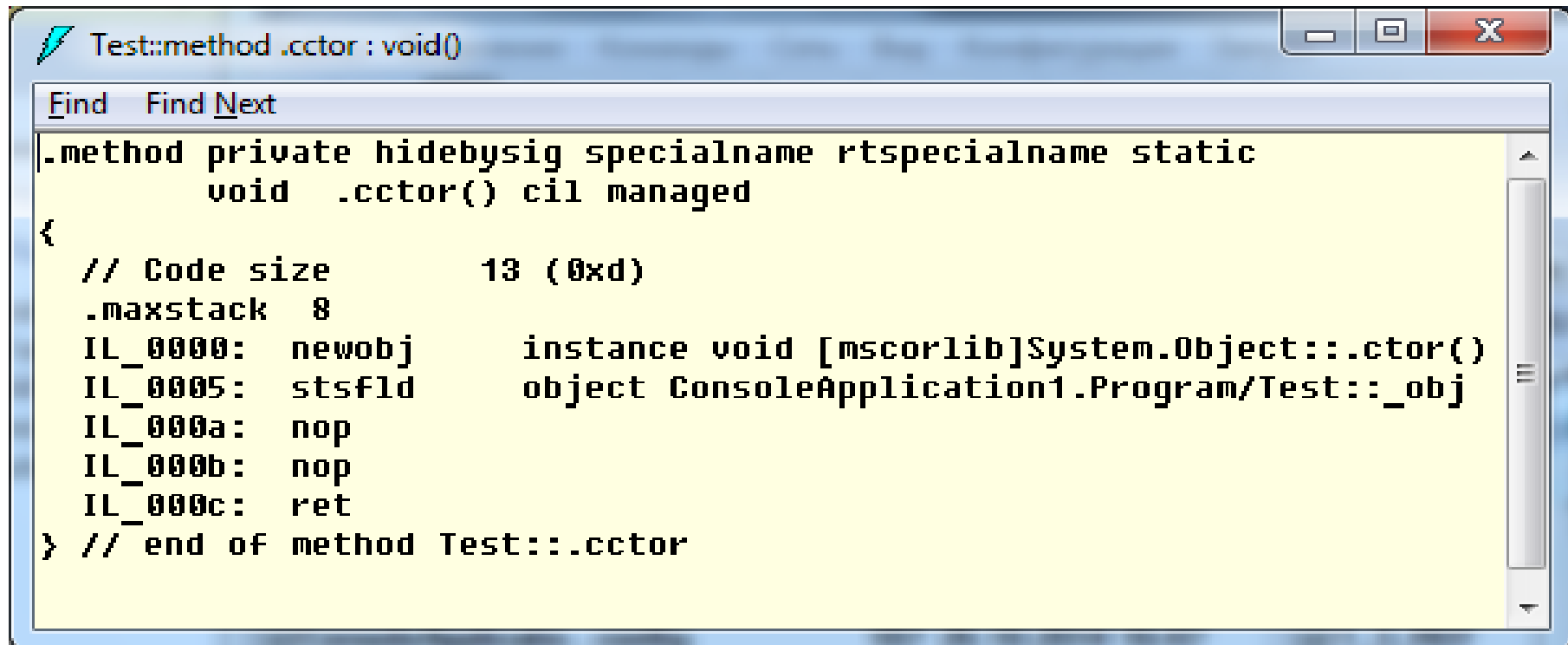
```
class Test
{
    static object _obj = new object();
    static Test() { }
}

static void Main()
{
    Test t = new Test();
}
```

## Создание статических типов и использование статических конструкторов

### LingPad 6.

```
class Test
{
    static object _obj = new object();
    static Test() { }
}
```



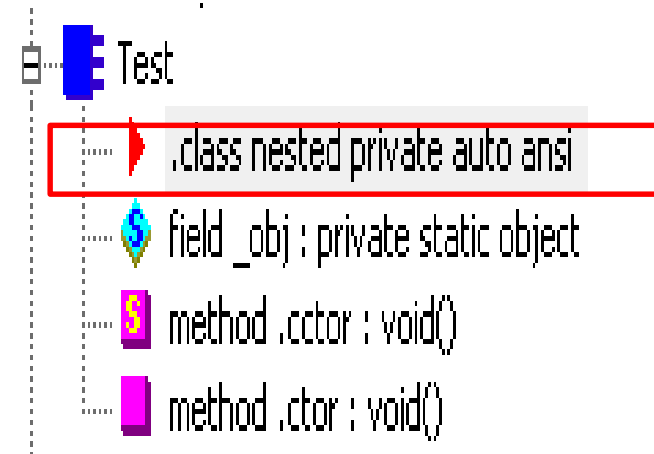
The screenshot shows a debugger window titled "Test::method .cctor : void()". The window contains a search bar with "Find" and "Find Next" buttons. Below the search bar, the IL code for the static constructor is displayed. The code is as follows:

```
.method private hidebysig specialname rtspecialname static
    void .cctor() cil managed
{
    // Code size          13 (0xd)
    .maxstack 8
    IL_0000: newobj          instance void [mscorlib]System.Object::.ctor()
    IL_0005: stsfld          object ConsoleApplication1.Program/Test::_obj
    IL_000a: nop
    IL_000b: nop
    IL_000c: ret
} // end of method Test::.cctor
```

## Создание статических типов и использование статических конструкторов

```
public class Test
{
    public static object _obj = new object();
}

static void Main()
{
    Test t = new Test();
}
```



```
.class auto ansi nested public beforefieldinit Test
    extends [mscorlib]System.Object
{
} // end of class Test
```

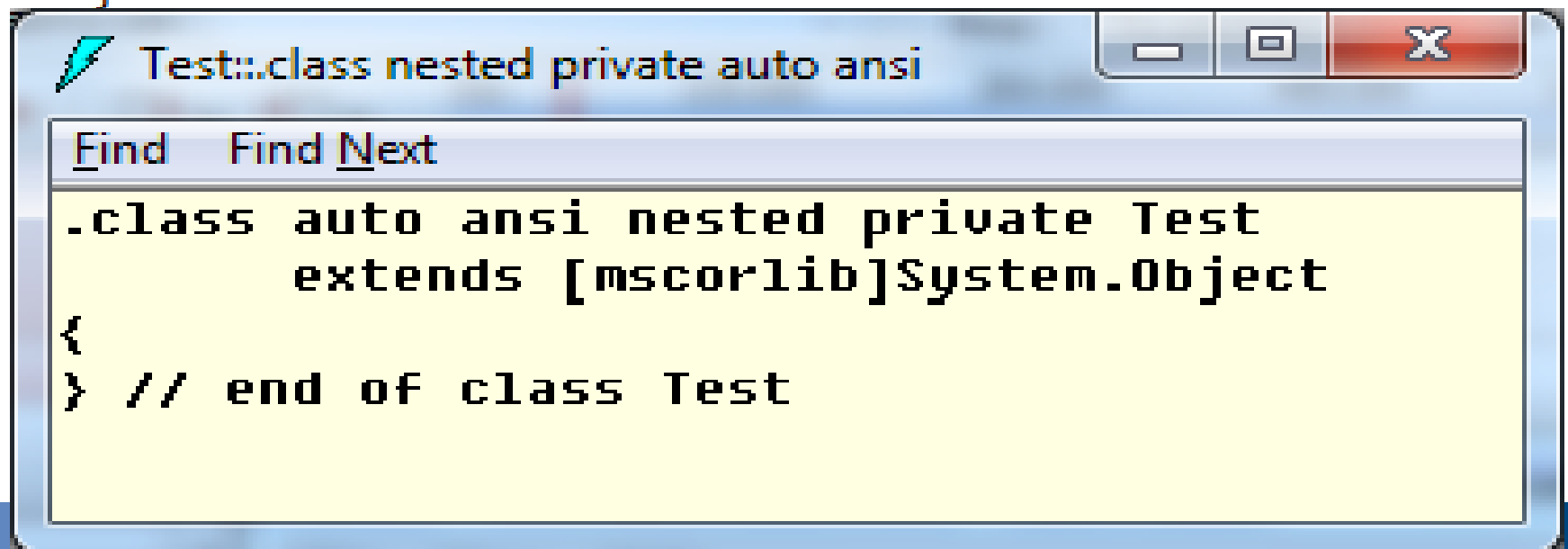


## Создание статических типов и использование статических конструкторов

```
class Test
{
    public static object _obj;

    static Test()
    {
        _obj = new object();
    }
}

static void Main()
{
    Test t = new Test();
}
```



# СПАСИБО ЗА ВНИМАНИЕ!

## ВОПРОСЫ?

NET.C#.04

Класс и Структуры

Конструктор и конструктор типа  
Singleton

**Author:** Саркисян Гаяне Феликсовна  
[gayane.f.sarkisyan@gmail.com](mailto:gayane.f.sarkisyan@gmail.com)