.NET Memory Management

Part1

Speaker: Ivan Kupriianov

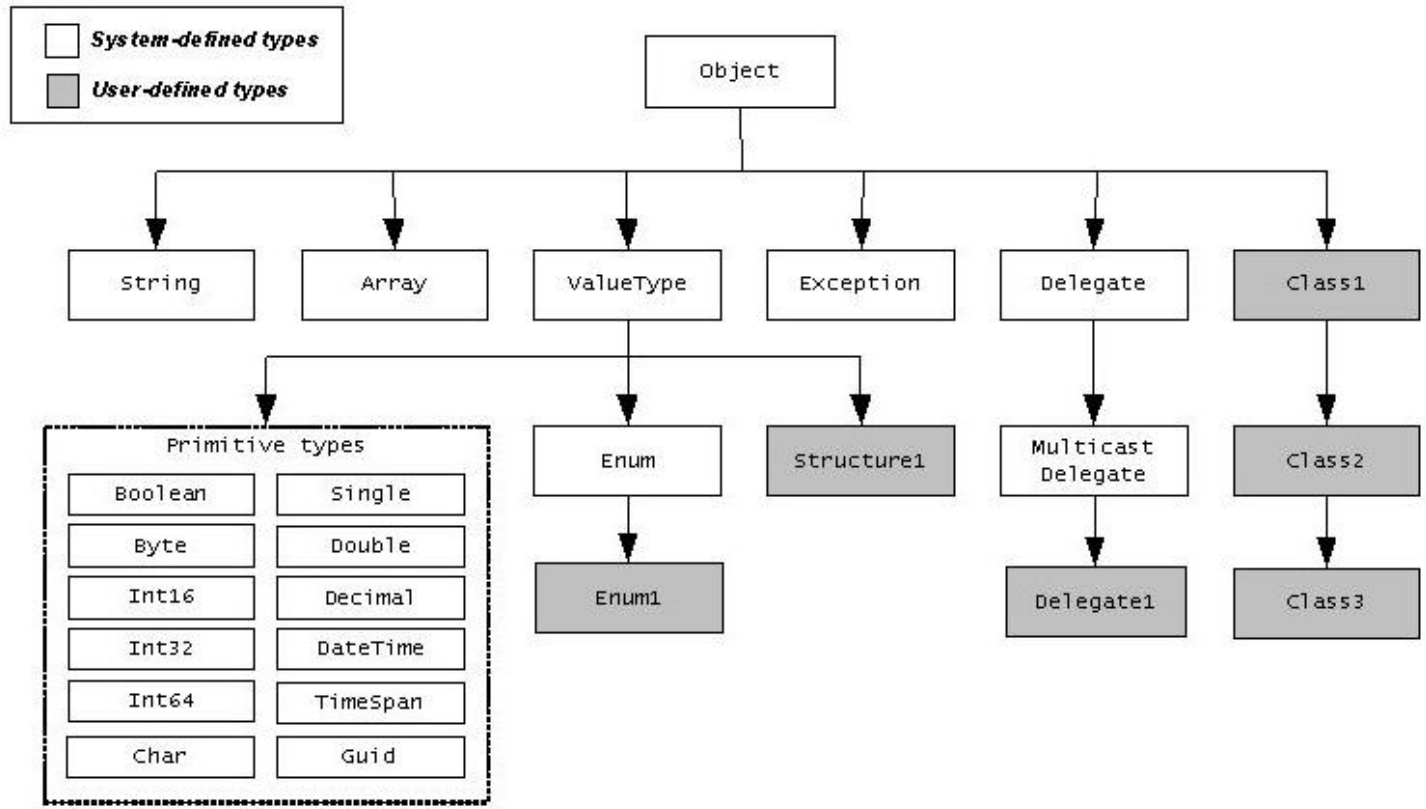
Position: .NET Software Engineer

Email: ivan_kupriianov@epam.com

Agenda

- Value & Reference types
- Operator new()
- Boxing & Unboxing

Types hierarchy according to Common Type System (CTS)



Value & Reference types. What's the difference?

Value types (Structures)	Reference types (Classes)
Implied to be immutable	Implied to be mutable
Variables directly contain data only	Variables contain a reference to the object
Memory allocated on the stack	Reference allocated on the stack; memory allocated on the heap
You cannot override parameterless ctor (default())	You can override one; you can declare class without it
Do not support inheritance, support interface implementation	Support both
Do not support finalizers	Support finalizers
You must initialize all the fields in ctor, if you declare one	You are free to avoid initialization

Struct initialization without calling new()

```
public struct MyStruct
{
    public int X;
    public int Y;

    public MyStruct(int x, int y)
    {
        X = x;
        Y = y;
    }
}
```

```
void Main()
{
    var s0 = new MyStruct();
    Console.WriteLine(s0);

    var s1 = new MyStruct(1, 1);
    Console.WriteLine(s1);

    MyStruct s2;
    Console.WriteLine(s2);

    MyStruct s3;
    s3.X = 1;
    s3.Y = 1;
    Console.WriteLine(s3);
}
```

What happens when new() operator called?

- Calculate memory in bytes

  Each instance field type bytes + base types bytes (e.g. System.Object) + type object pointer bytes + sync block index bytes
- Memory allocation in heap. All bytes set to 0.
- Sync block index and type object pointer init.
- Ctor call for current type (and base ones)
- Return reference for the newly created object.

And what about structs initialized with new()?

- Calculate memory in bytes

  Each instance field type bytes + base types bytes (e.g. System.Object).
- Memory allocation on stack. All bytes set to zero.
- Ctor call for current type (and base ones).
- Return reference for the newly created object.

```
// Reference type (because of 'class')
class  SomeRef { public Int32 x; }

// Value type (because of 'struct')
struct SomeVal { public Int32 x; }

static void ValueTypeDemo() {
   SomeRef r1 = new SomeRef();   // Allocated in heap
   SomeVal v1 = new SomeVal();   // Allocated on stack
   r1.x = 5;                     // Pointer dereference
   v1.x = 5;                     // Changed on stack
   Console.WriteLine(r1.x);      // Displays "5"
   Console.WriteLine(v1.x);      // Also displays "5"
   // The left side of Figure 5-2 reflects the situation
   // after the lines above have executed.

   SomeRef r2 = r1;              // Copies reference (pointer) only
   SomeVal v2 = v1;              // Allocate on stack & copies members
   r1.x = 8;                     // Changes r1.x and r2.x
   v1.x = 9;                     // Changes v1.x, not v2.x
   Console.WriteLine(r1.x);      // Displays "8"
   Console.WriteLine(r2.x);      // Displays "8"
   Console.WriteLine(v1.x);      // Displays "9"
   Console.WriteLine(v2.x);      // Displays "5"
   // The right side of Figure 5-2 reflects the situation
   // after ALL of the lines above have executed.
}
```

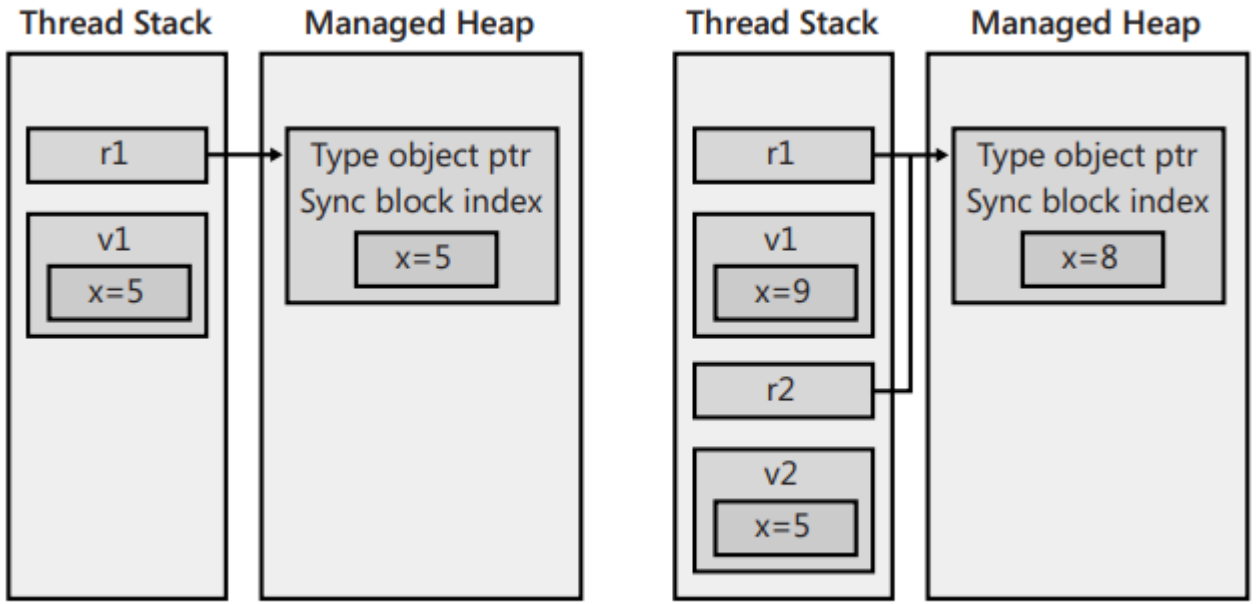


FIGURE 5-2 Visualizing the memory as the code executes

The picture is taken from CLR via C# by J. Richter

When to design a type as a struct over classes?

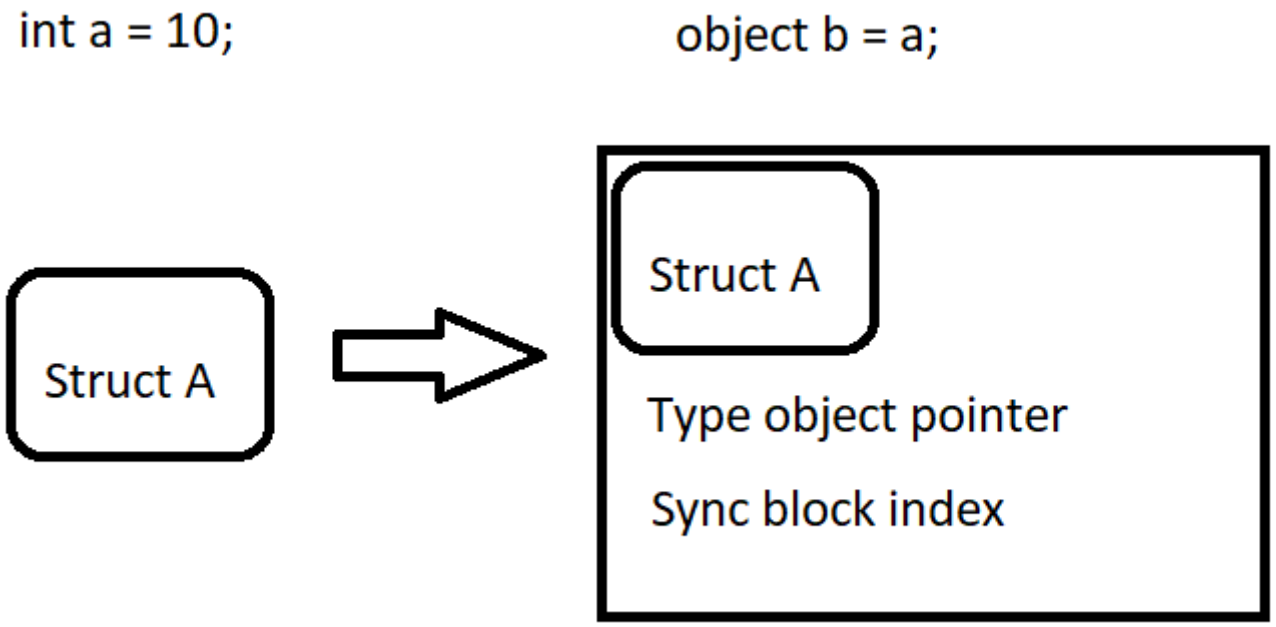
When an instance of the type:

- logically represents a single value, similar to primitive types (integer, double, etc.);
- has size smaller than 16 bytes;
- is immutable;
- will not have to be boxed frequently.

What is Sync block index and type object pointer?

- The type object pointer is a pointer to a type description of the object. This is used to find out what the actual type of an object is, for example needed to do virtual calls.
- The sync block index is an index in the table of synchronization blocks. Each object has a sync block, that contains information that serves several purposes in internals: storing object’s hash code, Thread Synchronization (Monitors), Garbage Collection Mark phase, GC finalization mechanism.

Boxing & unboxing



Expect boxing in all places. Even in those where you don’t expect it at all.

Where?

- ArrayList vs List<T> (Non-generic vs generic analogues)
- foreach
- Equals, GetType, GetHashCode, ToString (object instance methods), if they are not overridden

Consider code. What shows up in the console?

```
static void Main()
{
    var points = new List<Point>(Enumerable.Range(1, 10).Select(p => new Point()));
    foreach (var p in points)
    {
        p.IncX();
    }
    foreach (var p in points)
    {
        Console.WriteLine(p.X);
    }
}

public struct Point
{
    public int X;
    public void IncX()
    {
        X++;
    }
}
```

Solutions

- Change struct Point to class
- .ToArray() & increment foreach loop into for loop