Reflection it
Learn it better

Write C# Structs not (always) Classes

Fons Sonnemans

@fonssonnemans

Write C# Structs not (always) Classes

- Why
- C# 7.0 Ref Local/Return
- Public Fields not Properties
- Size
- Reference
  - Defensive Copy
    - Readonly fields, C# 7.2 in parameters, C# 7.2 ref readonly local/return
  - C# 7.2 readonly structs
  - C# 8.0 readonly members
- Equals() & GetHashCode()
- Boxing

Fons Sonnemans

- Software Development Consultant
  - Programming Languages
    - Clipper, Smalltalk, Visual Basic, C#
  - Platforms
    - Windows Forms, ASP.NET (Web Forms, MVC), XAML (WPF, Silverlight, Windows Phone, Windows 10)
  - Databases
    - MS SQL Server, Oracle
  - Role
    - Trainer, Coach, Advisor, Architect, Designer, App Developer
- More info: www.reflectionit.nl

Audience

- Who is using Visual Studio 2019
- Who is using C# 8.0
- Who is using .NET Core
- Who is writing structs regularly
- Who is writing methods with **ref** parameters
- Who has no performance issues
- Who checks there code on boxing & unboxing

Class versus Struct

Class

```
class Point {

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

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

    public void Swap() {
        (this.X, this.Y) = (this.Y, this.X);
    }

    public double Dist => Math.Sqrt((X * X) + (Y * Y));

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

Struct

```
struct Point {

    public int X;
    public int Y;

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

    public void Swap() {
        this = new Point(this.Y, this.X);
    }

    public double Dist => Math.Sqrt((X * X) + (Y * Y));

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

Why – Performance & Memory

```
class Program {
    static void Main(string[] args) => BenchmarkRunner.Run<BM>
}

[MemoryDiagnoser]
public class BM {

    [Benchmark]
    public void CreateClass() {
        for (int i = 0; i < 1000; i++) new PointClass(i, i);
    }

    [Benchmark(Baseline = true)]
    public void CreateStructs() {
        for (int i = 0; i < 1000; i++) new PointStruct(i, i);
    }
}
```

BenchmarkDotNet 0.12.0

Powerful .NET library for benchmarking

Package Manager | .NET CLI | PackageReference | Paket CLI

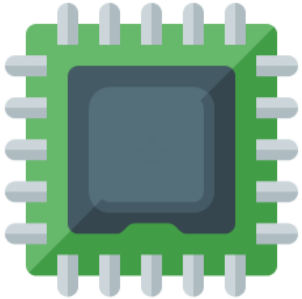
```
PM> Install-Package BenchmarkDotNet -Version 0.12.0
```

```
BenchmarkDotNet=v0.12.0, OS=Windows 10.0.19037
Intel Core i7-2600K CPU 3.40GHz (Sandy Bridge), 1 CPU, 8 logical and 4 physical cores
.NET Core SDK=3.1.100
  [Host]     : .NET Core 3.1.0 (CoreCLR 4.700.19.56402, CoreFX 4.700.19.56404), X64 RyuJIT
  DefaultJob : .NET Core 3.1.0 (CoreCLR 4.700.19.56402, CoreFX 4.700.19.56404), X64 RyuJIT
```

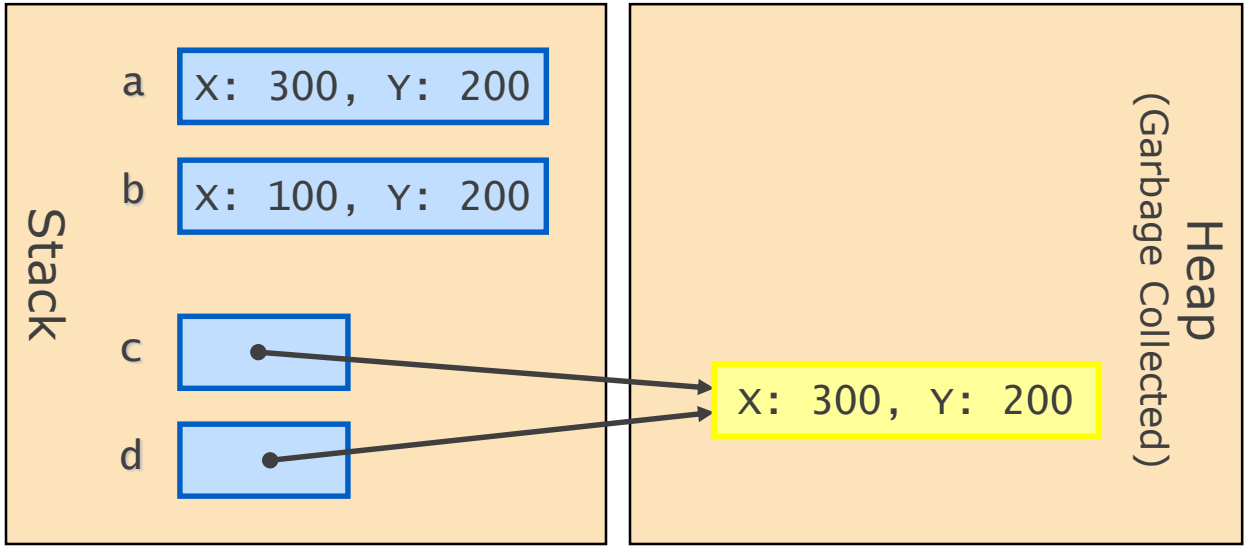
Method	Mean	Error	StdDev	Ratio	RatioSD	Gen 0	Gen 1	Gen 2	Allocated
CreateClass	4,611.9 ns	57.18 ns	50.69 ns	18.98	0.19	5.7373	-	-	24000 B
CreateStructs	243.2 ns	1.23 ns	1.15 ns	1.00	0.00	-	-	-	-

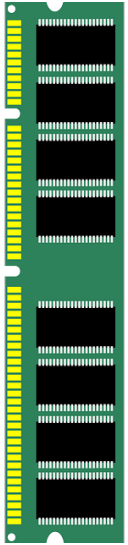
Value Types vs. Reference Types

```
PointStruct a = new PointStruct(100, 200);
PointStruct b = a; // Copy
a.X = 300;

PointClass c = new PointClass(100, 200);
PointClass d = c;
c.X = 300;
```



"CPU Cache"
L1 -> L2 -> L3

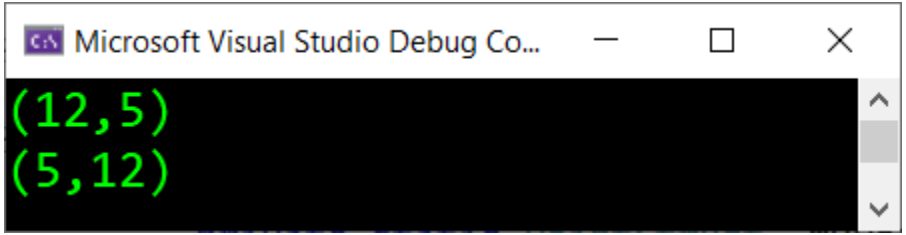




"Memory"

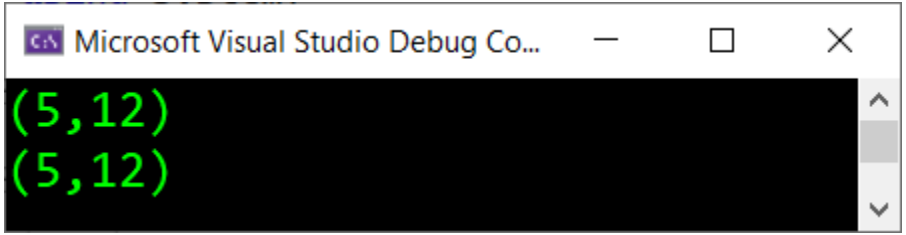
Copy

```
class Program {

    private static PointStruct _myPoint = new PointStruct(12, 5);

    static void Main(string[] args) {
        var p = GetPoint();
        p.Swap();

        Console.WriteLine(_myPoint.ToString());
        Console.WriteLine(p.ToString());
    }

    public static PointStruct GetPoint() { // Returns a copy
        return _myPoint;
    }
}
```



C# 7.0 - Ref Local & Return

```
class Program {

    private static PointStruct _myPoint = new PointStruct(12, 5);

    static void Main(string[] args) {
        ref var p = ref GetPoint();
        p.Swap();

        Console.WriteLine(_myPoint.ToString());
        Console.WriteLine(p.ToString());
    }

    public static ref PointStruct GetPoint() { // Returns an Alias/Shortcut
        return ref _myPoint;
    }
}
```



Why Mutable structs should use Fields and not Properties

```
class Program {
    static void Main(string[] args) {
        LineStruct l = new LineStruct(new PointStruct(1, 5),
                                      new PointStruct(8, 5));

        Console.WriteLine(l.ToString());
        Console.WriteLine(l.Length.ToString());

        l.Start.Swap();

        Console.WriteLine(l.ToString());
        Console.WriteLine(l.Length.ToString());
    }
}
```

C:\Windows\system32\cmd.exe

```
Line (1,5) to (8,5)
7
Line (1,5) to (8,5)
7
Press any key to continue . . .
```

LineStruct
Struct

- Properties
  - Length { get; } : double
  - Start { get; set; } : PointStruct
  - Stop { get; set; } : PointStruct
- Methods
  - LineStruct(PointStruct start, PointStruct stop)
  - ToString() : string

PointStruct
Struct

- Properties
  - X { get; set; } : int
  - Y { get; set; } : int
- Methods
  - PointStruct(int x, int y)
  - Swap() : void
  - ToString() : string

Why Mutable structs should use Fields and not Properties

```
class Program {
    static void Main(string[] args) {
        LineStruct l = new LineStruct(new PointStruct(1, 5),
                                      new PointStruct(8, 5));

        Console.WriteLine(l.ToString());
        Console.WriteLine(l.Length.ToString());

        l.Start.Swap();

        Console.WriteLine(l.ToString());
        Console.WriteLine(l.Length.ToString());
    }
}
```

C:\Windows\system32\cmd.exe

```
Line (1,5) to (8,5)
7
Line (5,1) to (8,5)
5
Press any key to continue . . .
```

LineStruct
Struct

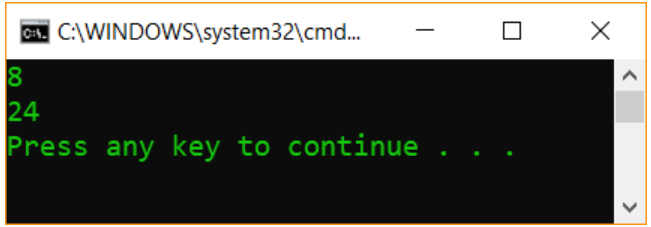
- Fields
  - Start : PointStruct
  - Stop : PointStruct
- Properties
  - Length { get; } : double
- Methods
  - LineStruct(PointStruct start, PointStruct stop)
  - ToString() : string

PointStruct
Struct

- Fields
  - X : int
  - Y : int
- Methods
  - PointStruct(int x, int y)
  - Swap() : void
  - ToString() : string

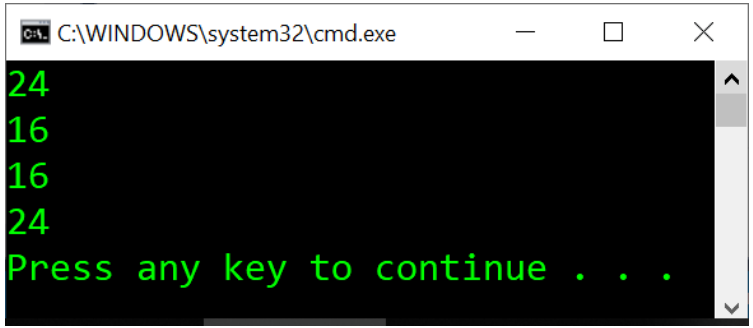
Size

```
class Program {
    static void Main(string[] args) {
        // Install-Package System.Runtime.CompilerServices.Unsafe

        var size = Unsafe.SizeOf<PointStruct>();
        Console.WriteLine(size);

        size = Unsafe.SizeOf<Test>();
        Console.WriteLine(size);
    }
}

struct Test {
    public int A;
    public long C;
    public bool B;
}

struct PointStruct {
    public int X, Y;
}
```



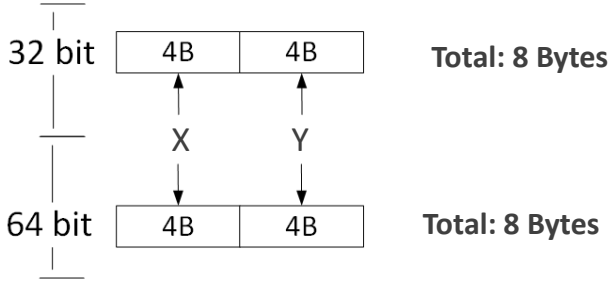
Padding & StructLayout

```
internal class Program {
    private static void Main(string[] args) {
        // Install-Package System.Runtime.CompilerServices.Unsafe
        Console.WriteLine(Unsafe.SizeOf<Test1>());
        Console.WriteLine(Unsafe.SizeOf<Test2>());
        Console.WriteLine(Unsafe.SizeOf<Test3>());
        Console.WriteLine(Unsafe.SizeOf<Test3?>());    }
}

internal struct Test1 { // 24 bytes
    public int A;  // 4 bytes + 4 Padding
    public long B; // 8 bytes
    public bool C; // 1 byte + 7 padding
}

internal struct Test2 { // 16 bytes
    public int A;  // 4 bytes
    public bool C; // 1 byte + 3 padding
    public long B; // 8 bytes
}

[StructLayout(LayoutKind.Auto)]
internal struct Test3 { // 16 bytes (B,A,C + 3 padding)
    public int A;
    public long B;
    public bool C;
}
```

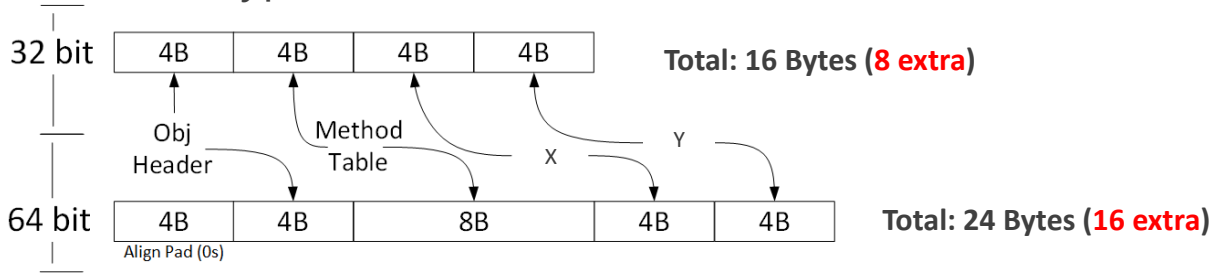


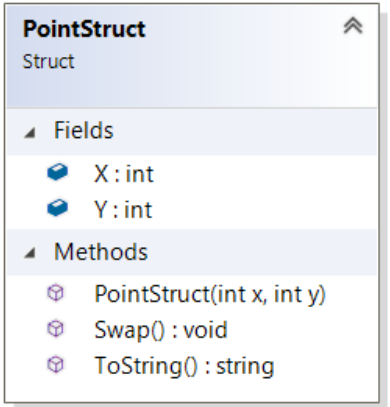
Value Types vs. Reference Types – Memory Overhead

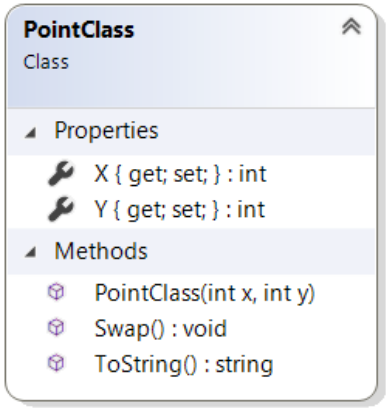
- Value Type (PointStruct)



- Reference Type (PointClass)







- Source
  - http://adamsitnik.com/Value-Types-vs-Reference-Types/

C# 1.0: Defensive Copy of **Readonly Struct Fields**

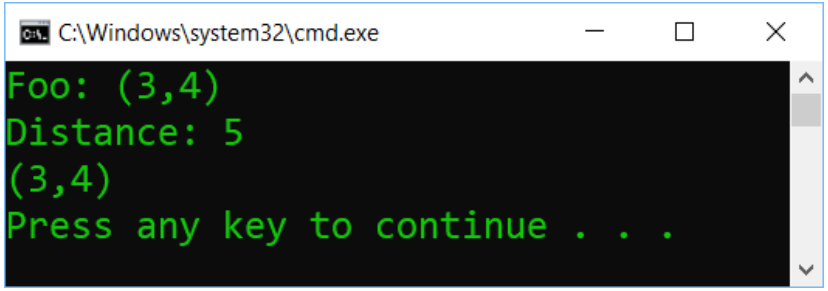
```
class Program {

    static readonly PointStruct _p = new PointStruct(3, 4);

    static void Main(string[] args) {

        Console.WriteLine(_p.ToString());

        // Allowed but doesn't do anything, it is executed on a defensive copy !!!
        _p.Swap();

        Console.WriteLine(_p.ToString());

        //_p.X = 100; Won't compile, _p is readonly
    }
}
```

C:\Windows\system32\cmd.exe

```
(3,4)
(3,4)
Press any key to continue . . .
```

C# 7.2 - **in** parameters

```
class Program {
    static void Main(string[] args) {
        PointStruct pnt = new PointStruct(3, 4);
        Foo(pnt);
        Console.WriteLine(pnt.ToString());
    }

    private static void Foo(in PointStruct p) { // Alias/shortcut ("Cheap")
        // Allowed but doesn't do anything, it is executed on a defensive copy !!!
        p.Swap();

        // Allowed but also executed on another defensive copy !!!
        Console.WriteLine("Foo: " + p.ToString());

        // Allowed but also executed on another defensive copy !!!
        var dist = p.Dist;
        Console.WriteLine("Distance: " + dist.ToString());
    }
```



C# 7.2 - **in** parameters

SharpLab

https://sharplab.io/#v2:EYLgtghgzgLgpgJwD4AEAMACFBGA3AWACh0tsAWA4zHAOgBkBLAOwEdKTaAVOADxnerYArJSJMIYOFAAOEAMZwMAYQD2TKCoA2cAILTpZAMwYA3kQwWsAJlIB2U

Code C# master (1 Dec 2017)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp43 {
    class Program {
        static void Main(string[] args) {
            PointStruct p3 = new PointStruct(3, 4);
            Foo(p3);
            Console.WriteLine(p3);
        }

        private static void Foo(in PointStruct p) { // Reference = Pointer ("
            p.Swap();
            Console.WriteLine("Foo: " + p.ToString());

            var dist = p.Dist;
            Console.WriteLine("Distance: " + dist);
        }

    }
```

Branch master, last commit

37ef0cdea72622fa9195b770808624c8dca007b3 at 01 Dec 2017 by Sam Harwell
Merge pull request #23491 from sharwell/optimize-isdefinedinsourcetree

Avoid allocations in IsDefinedInSourceTree hot paths

Results C# Release

```
{
    internal class Program
    {
        private static void Main(string[] args)
        {
            PointStruct pointStruct = new PointStruct(3L, 4L);
            Program.Foo(ref pointStruct);
            Console.WriteLine(pointStruct);
        }

        private static void Foo([IsReadOnly] [In] ref PointStruct p)
        {
            PointStruct pointStruct = p;
            pointStruct.Swap();
            string arg_27_0 = "Foo: ";
            pointStruct = p;
            Console.WriteLine(arg_27_0 + pointStruct.ToString());
            pointStruct = p;
            double dist = pointStruct.Dist;
            Console.WriteLine("Distance: " + dist);
        }
    }
    internal struct PointStruct
    {
        [CompilerGenerated]
        private long <X>k__BackingField;

        [CompilerGenerated]
        private long <Y>k__BackingField;

        public long X
```

Defensive Copy

Defensive Copy

Built by Andrey Shchekin (@ashmind) — see SharpLab on GitHub.

C# 7.2 – ref **readonly** locals & returns

```
class Program {

    private static PointStruct _myPoint = new PointStruct(12, 5);

    static void Main(string[] args) {
        ref readonly var p = ref GetPoint();
        p.Swap();

        Console.WriteLine(_myPoint.ToString());
        Console.WriteLine(p.ToString());
    }

    public static ref readonly PointStruct GetPoint() { // Returns an Alias
        return ref _myPoint;
    }
}
```

Microsoft Visual Studio Debug Console

```
(12,5)
(12,5)
```

C# 7.2 – ref **readonly** locals & returns

SharpLab

https://sharplab.io/#v2:CYLg1APgAgTAjAWAFDIHYEMC2BTAzgB3QGNsACAJWwDNKAXAVwCdUBBVYSqgGQHsj0ANqQDeyUuNKxJcAOwjkYiUvyMAIgD

Code C# Create Gist Default

```c#
using System;

namespace RefReturnAndRefLocal {
    class Program {

        private static PointStruct _myPoint = new PointStruct(12, 5);

        static void Main(string[] args) {
            ref readonly var p = ref GetPoint();
            p.Swap();

            Console.WriteLine(_myPoint.ToString());
            Console.WriteLine(p.ToString());
        }

        public static ref readonly PointStruct GetPoint() { // Returns
            return ref _myPoint;
        }
    }

    struct PointStruct {

        public int X; // { get; set; }
        public int Y; // { get; set; }

        public PointStruct(int x, int y) {
            this.X = x;
            this.Y = y;
        }

        public void Swap() {
            this = new PointStruct(this.Y, this.X);
        }
```

Results C# Debug

```c#
using System;
using System.Diagnostics;
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Security;
using System.Security.Permissions;

[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: Debuggable(DebuggableAttribute.DebuggingModes.Default | Debugg
[assembly: SecurityPermission(SecurityAction.RequestMinimum, SkipVerifica
[assembly: AssemblyVersion("0.0.0.0")]
[module: UnverifiableCode]
namespace RefReturnAndRefLocal
{
    internal class Program
    {
        private static PointStruct _myPoint = new PointStruct(12, 5);

        private static void Main(string[] args)
        {
            ref PointStruct point = ref GetPoint();
            PointStruct pointStruct = point;
            pointStruct.Swap();
            Console.WriteLine(_myPoint.ToString());
            pointStruct = point;
            Console.WriteLine(pointStruct.ToString());
        }

        [return: IsReadOnly]
        public static ref PointStruct GetPoint()
        {
            return ref _myPoint;
```

Defensive Copy

Defensive Copy

Theme: Auto | Built by Andrey Shchekin (@ashmind) — see SharpLab on GitHub.

Avoiding Defensive copies

- C# 7.2 – readonly structs

```
readonly struct PointStruct {

    public readonly int X, Y;

    public PointStruct Swap() => new PointStruct(this.Y, this.X);
```

- C# 8.0 – readonly members

```
struct PointStruct {

    public int X, Y;

    public readonly double Dist => Math.Sqrt((X * X) + (Y * Y));

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

C# 7.2 – readonly structs

SharpLab

https://sharplab.io/#v2:CYLg1APgAgTAjAWAFDIHYEMC2BTAzgB3QGNsACAJWwDNKAXAVwCdUBBVYSqgGQHsj0ANqQDeyUuNKxJcAOwjkYiUvyMAlgD

Code C# Create Gist Default

```
using System;

namespace RefReturnAndRefLocal {
    class Program {

        private static PointStruct _myPoint = new PointStruct(12, 5);

        static void Main(string[] args) {
            ref readonly var p = ref GetPoint();
            Console.WriteLine(p.Swap());
            Console.WriteLine(_myPoint.ToString());
            Console.WriteLine(p.ToString());
        }

        public static ref readonly PointStruct GetPoint() { // Returns
            return ref _myPoint;
        }
    }

    readonly struct PointStruct {

        public readonly int X;
        public readonly int Y;

        public PointStruct(int x, int y) {
            this.X = x;
            this.Y = y;
        }

        public PointStruct Swap() => new PointStruct(this.Y, this.X);

        public double Dist => Math.Sqrt((X * X) + (Y * Y));
```

Results C# Debug

```
using System;
using System.Diagnostics;
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Security;
using System.Security.Permissions;

[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: Debuggable(DebuggableAttribute.DebuggingModes.Default | Debugg
[assembly: SecurityPermission(SecurityAction.RequestMinimum, SkipVerifica
[assembly: AssemblyVersion("0.0.0.0")]
[module: UnverifiableCode]
namespace RefReturnAndRefLocal
{
    internal class Program
    {
        private static PointStruct _myPoint = new PointStruct(12, 5);

        private static void Main(string[] args)
        {
            ref PointStruct point = ref GetPoint();
            Console.WriteLine(point.Swap());
            Console.WriteLine(_myPoint.ToString());
            Console.WriteLine(point.ToString());
        }

        [return: IsReadOnly]
        public static ref PointStruct GetPoint()
        {
            return ref _myPoint;
        }
    }
```

C# 8.0 – readonly members

SharpLab

https://sharplab.io/#v2:CYLg1APgAgTAjAWAFDIHYEMC2BTAzgB3QGNsACAJWwDNKAXAVwCdUBBVYSqgGQHsj0ANqQDeyUuNKxJcAOwjkYiUvyMAlgD

Code C# Create Gist Default

```
class Program {

    private static PointStruct _myPoint = new PointStruct(12, 5);

    static void Main(string[] args) {
        ref readonly var p = ref GetPoint();
        Console.WriteLine(p.Swap());
        Console.WriteLine(_myPoint.ToString());
        Console.WriteLine(p.ToString());
    }

    public static ref readonly PointStruct GetPoint() { // Returns
        return ref _myPoint;
    }
}

struct PointStruct {

    public int X, Y;

    public PointStruct(int x, int y) {
        this.X = x;
        this.Y = y;
    }

    public readonly PointStruct Swap() => new PointStruct(this.Y, t

    public readonly double Dist => Math.Sqrt((X * X) + (Y * Y));

    public readonly override string ToString() => $"({X.ToString()}
}

}
```

Results C# Debug

```
using System;
using System.Diagnostics;
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Security;
using System.Security.Permissions;

[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: Debuggable(DebuggableAttribute.DebuggingModes.Default | Debugg
[assembly: SecurityPermission(SecurityAction.RequestMinimum, SkipVerifica
[assembly: AssemblyVersion("0.0.0.0")]
[module: UnverifiableCode]
namespace RefReturnAndRefLocal
{
    internal class Program
    {
        private static PointStruct _myPoint = new PointStruct(12, 5);

        private static void Main(string[] args)
        {
            ref PointStruct point = ref GetPoint();
            Console.WriteLine(point.Swap());
            Console.WriteLine(_myPoint.ToString());
            Console.WriteLine(point.ToString());
        }

        [return: IsReadOnly]
        public static ref PointStruct GetPoint()
        {
            return ref _myPoint;
        }
    }
```

ReflectionIT.Analyzer.Structs (VS2019 Only)

```
0 references
internal class Program { // .NET Core 2.2 project !

    0 references
    private static void Main(string[] args) {
        int a = 5;
        ref readonly int b = ref a;
    }

    0 references
    private static void Foo(in double a, in DateTime b, in Point c)
    }
}

1 reference
struct Point {
    public readonly int X, Y;
}
```

Code	Description	Project	File	..	Suppression State
RAS0003	Ref readonly local 'Int32' is a non-readonly struct which may leads to defensive copies	ConsoleApp165	Program.cs	8	Active
RAS0001	In parameter 'a' is a non-readonly struct which may leads to defensive copies	ConsoleApp165	Program.cs	11	Active
RAS0001	In parameter 'c' is a non-readonly struct which may leads to defensive copies	ConsoleApp165	Program.cs	11	Active
RAS0002	Readonly field 'X' is a non-readonly struct which may leads to defensive copies	ConsoleApp165	Program.cs	16	Active
RAS0002	Readonly field 'Y' is a non-readonly struct which may leads to defensive copies	ConsoleApp165	Program.cs	16	Active

- https://www.nuget.org/packages/ReflectionIT.Analyzer.Structs/

.NET Core 3.0

- All primitives (int, double, decimal, long, DateTime, Timespan, etc) are now **readonly**
- Most mutable structs have now **readonly members**

```
internal class Program { // .NET Core 3.0 project !

    private static void Main(string[] args) {
        int a = 5;
        ref readonly int b = ref a;
    }

    private static void Foo(in double a, in DateTime b, in Point c)
    }
}

readonly struct Point {
    public readonly int X, Y;
}
}
```

== and Equals

- The Equals method is just a virtual one defined in System.Object, and overridden by whichever classes choose to do so. The == operator is an operator which can be overloaded by classes, but which usually has identity behaviour.
- For reference types where == has not been overloaded, it compares whether two references refer to the same object - which is exactly what the implementation of Equals does in System.Object.
- Value types do not provide an overload for == by default. However, most of the value types provided by the framework provide their own overload.
- The default implementation of Equals for a value type is provided by **ValueType**, and uses **reflection** to make the comparison, which makes it significantly slower than a type-specific implementation normally would be.

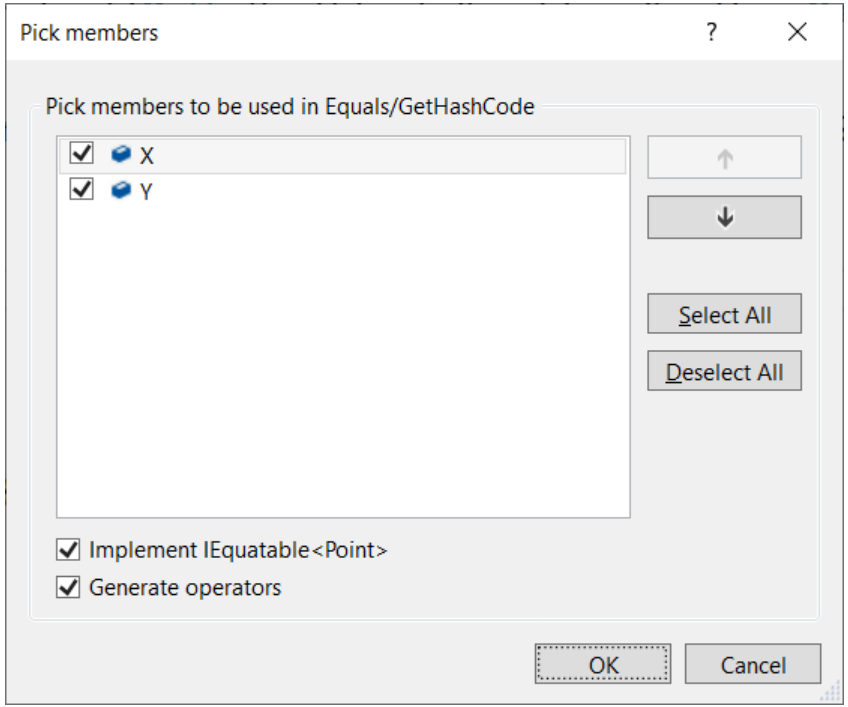
You should always re-implement Equals(), GetHashCode() and implement IEquatable<T> for your structs, because the default implementations uses boxing and reflection to enumerate over the fields.

Generate Equals and GetHashCode…

```
10 references
43  struct Point {
44
45      int X, Y;
46
47      Point(int x, int y) {
48          X = x;
49          Y = y;
50
51
        1 reference
52      public void Swap() {
53          this = new Point(Y, X);
54      }
```

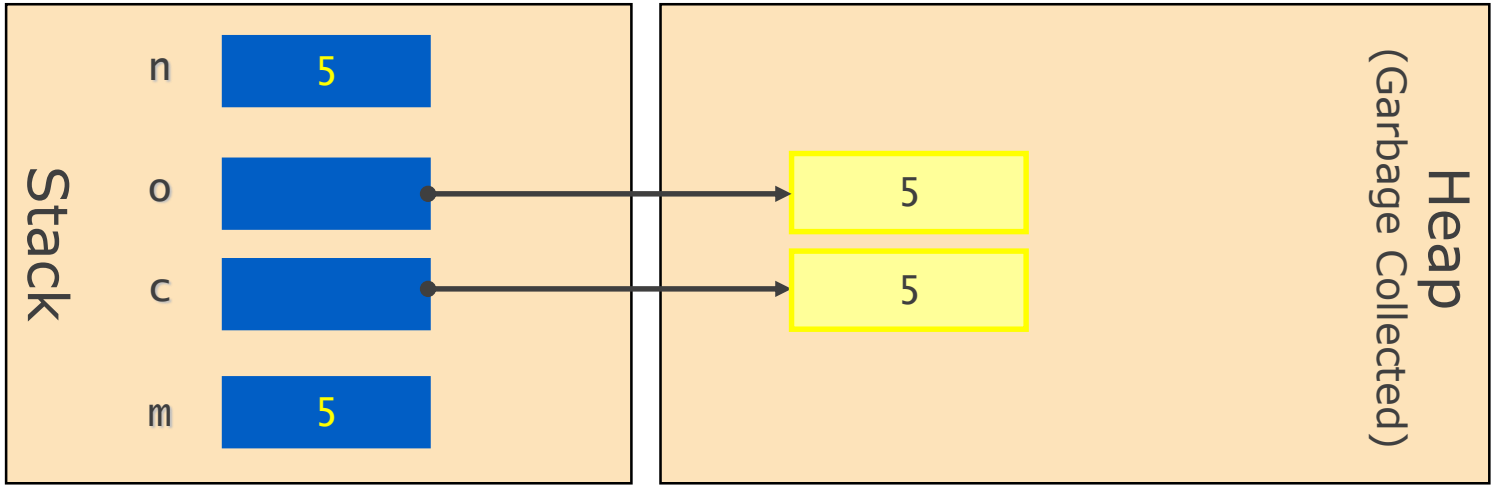
- Sort Members
- Generate constructor...
- Generate Equals(object)...
- Generate Equals and GetHashCode...
- Generate overrides...
- Extract Interface...
- Move type to Point.cs
- Add accessibility modifiers

This will make Dictionaries, HashSets and LINQ faster



Boxing and Unboxing

```
int n = 5;
object o = n;              // boxing, moving to the Heap
IComparable c = n;         // boxing, moving to the Heap
int m = (int)o;            // Unboxing, moving back to the Stack
```



ClrHeapAllocationAnalyzer

ConsoleApp71 - Microsoft Visual Studio

```
0 references
private static void Main(string[] args) {

    var l = new List<Employee>(100);
    for (int i = 0; i < l.Count; i++) {
        l.Add(new Employee("Fons" + i, i));
    }

    double d = 12.5;

    var s = string.Format("Value {0:C}", d);

}
```

Error List

Code	Description	Line	File
HeapAnalyzerStringConcatRule	Considering using StringBuilder	13	Program.cs
HeapAnalyzerBoxingRule	Value type (int) is being boxed to a reference type for a string concatenation.	13	Program.cs
HeapAnalyzerBoxingRule	Value type to reference type conversion causes boxing at call site (here), and unboxing at the callee-site. Consider using generics if applicable	18	Program.cs

Solution Explorer

- Solution 'ConsoleApp71' (1 project)
  - ConsoleApp71
    - Properties
    - References
      - Analyzers
        - ClrHeapAllocationAnalyzer
      - Microsoft.CSharp
      - System
      - System.Core
      - System.Data
      - System.Data.DataSetExtensions
      - System.Net.Http
      - System.Xml
      - System.Xml.Linq
    - App.config
    - Program.cs

Package Manager | .NET CLI | Paket CLI

```
PM> Install-Package ClrHeapAllocationAnalyzer -Version 1.0.0.8
```

Summary

- Use structs if performance & memory is an issue and you don't need inheritance
  - Structs always have a parameterless contructor
- Use Fields not Properties
- If you have more than 2 fields with different length use the StructLayout attribute
- Make your structs immutable (readonly)
  - Mutable structs should have readonly members
- Avoid boxing & unboxing

- Use analyzers to check your code

@fonssonnemans

fons.sonnemans@reflectionit.nl

fonssonnemans

reflectionit.nl/blog

github.com/sonnemaf

Copyright