Todd Howard: "If you're running low on memory, you can reboot the original Xbox and the user can't tell. You can throw a screen up. When Morrowind loads sometimes you get a very long load. That's us rebooting the Xbox."

Why did the Java developer wear glasses?

Because he couldn't see sharp.

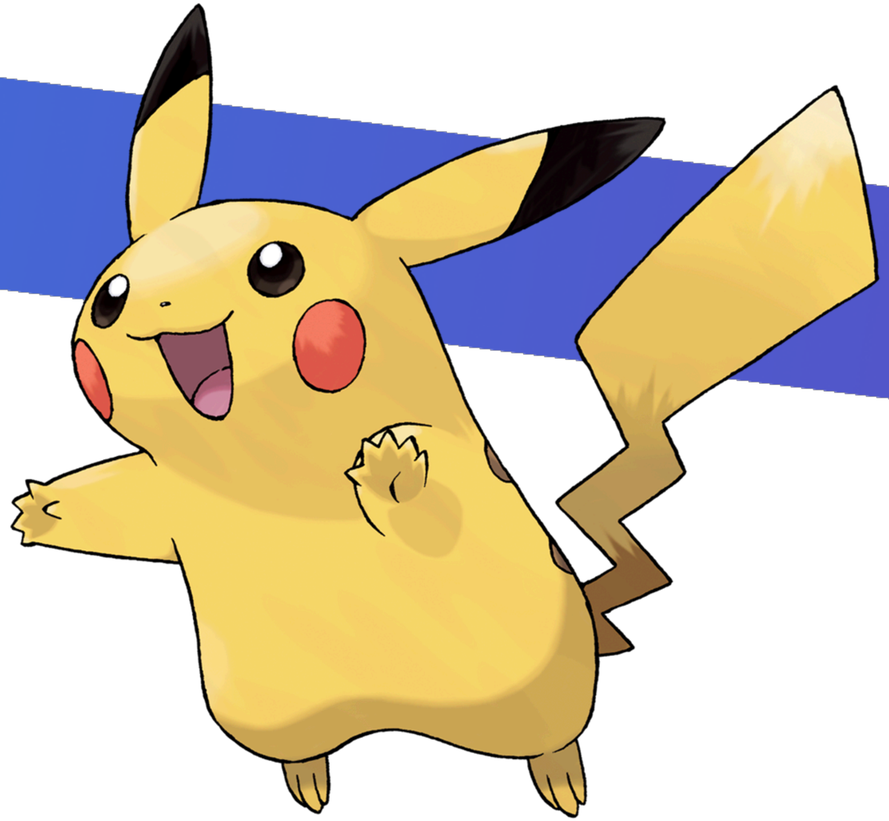
C# Lowering

Was ist das und warum muss mich das interessieren?

Steven Giesel // DWX 2024 // Wie man sharplab.io für einen kompletten Talk missbrauchen kann.

```
{
  "fullName": "Steven Giesel",
  "websites": [
    "https://steven-giesel.com",
    "https://giesel.engineering"
  ],
  "canHire": true,
  "isMvp": true,
  "linkedIn": true,
  "x": false
}
```


Was haben diese beiden gemeinsam?



```
foreach (var name in names)
{
    ...
}
```

Keiner von beiden ist im IL-Code bekannt

Motivation

"*Understand one level below your normal abstraction layer.*" -Neal Ford

Performance

Motivation

"*Understand one level below your normal abstraction layer.*" -Neal Ford

Performance

Bugs

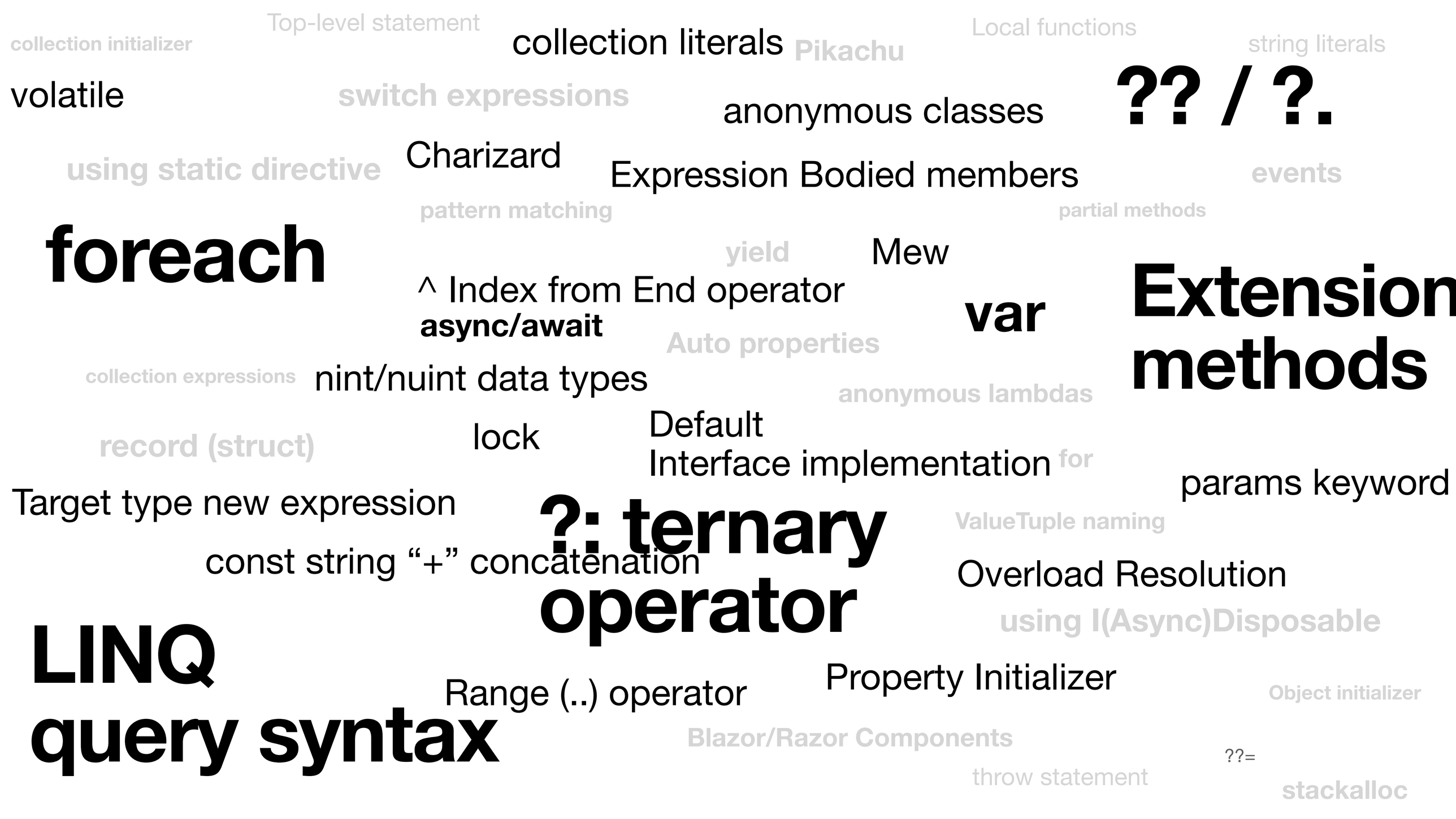
Motivation

"*Understand one level below your normal abstraction layer.*" -Neal Ford

Performance

Bugs

Grundlagen


Top-level statement
collection initializer
collection literals
Pikachu
Local functions
string literals
volatile
switch expressions
anonymous classes
?? / ?.
Charizard
using static directive
Expression Bodied members
events
pattern matching
partial methods
foreach
yield
Mew
^ Index from End operator
async/await
var
Extension methods
Auto properties
collection expressions
nint/nuint data types
anonymous lambdas
record (struct)
lock
Default Interface implementation
for
params keyword
Target type new expression
?: ternary operator
ValueTuple naming
const string “+” concatenation
Overload Resolution
using I(Async)Disposable
LINQ query syntax
Property Initializer
Range (..) operator
Object initializer
Blazor/Razor Components
??=
throw statement
stackalloc

Und das allerwichtigste keyword von allen:

dynamic

Was ist “Lowering”?

Was ist “Lowering”?

Compiling

Übersetzen einer Sprache in eine maschinennähere Sprache

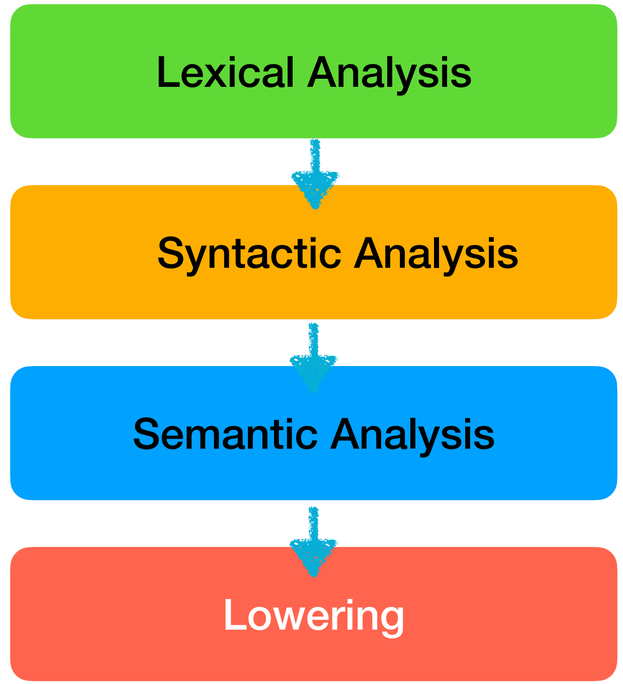
IL
(Intermediate Language)

Was ist “Lowering”?

Lowering

Compiling

Was ist “Lowering”?



- Ein andere Name dafür ist oft “syntactic sugar”
- Oder “compiler magic”
- Lowering ist Teil des Kompilerprozesses, wenn C# Code zu IL-Code (oder direkt zu Maschinencode) übersetzt wird

Dank NativeAOT

Vorteile die “Lowering” bringt

Optimierung

Vorteile die “Lowering” bringt

Optimierung

Einfachheit

Vorteile die “Lowering” bringt

Optimierung

Einfachheit

Kompatibilität
Konsistenz

Vorteile die “Lowering” bringt

Optimierung

Einfachheit

Kompatibilität
Konsistenz

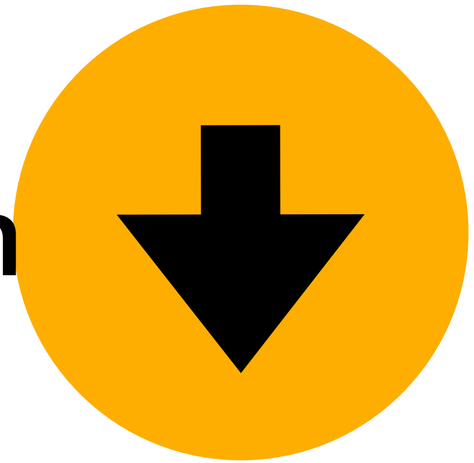
Kompiler

Was ist “Lowering”?

Fliegen

Was ist “Lowering”?

Schnell durch die Luft bewegen

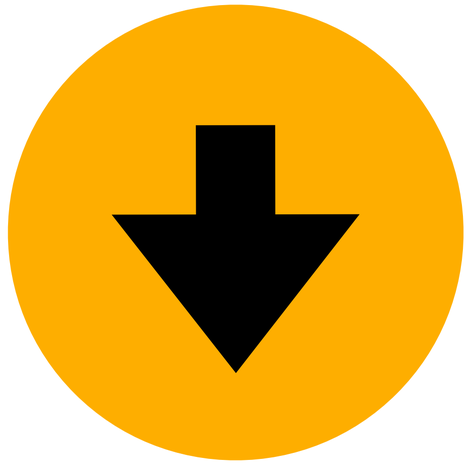


Was ist “Lowering”?

Horizont

Was ist “Lowering”?

Grenze zwischen Himmel und Erde




<code/>

Fangen wir einfach an - var

```
var myString = "Hello World";

Console.Write(myString);
```

Wird “lowered” zu

```
string myString = "Hello World";

Console.Write(myString);
```

- **var** “existiert” nicht und wird zum konkreten Typ aufgelöst
- Dies nennt sich: **type interference** (die Fähigkeit, den Typ aus dem Kontext abzuleiten)

Was ist die Ausgabe des folgenden Codes?

```
var entity = new MyEntity();

Console.Write($"{entity.GetCounter}, ");
Console.Write($"{entity.GetCounter}, ");
Console.Write($"{entity.ExprCounter}, ");
Console.Write($"{entity.ExprCounter}");


public class MyEntity
{
  private static int a = 0;
  private static int b = 0;

  public int GetCounter { get; } = ++a;
  public int ExprCounter => ++b;
}
```

Was ist die Ausgabe des folgenden Codes?

```
var entity = new MyEntity();

Console.Write($"{entity.GetCounter}, ");
Console.Write($"{entity.GetCounter}, ");
Console.Write($"{entity.ExprCounter}, ");
Console.Write($"{entity.ExprCounter}");


public class MyEntity
{
  private static int a = 0;
  private static int b = 0;

  public int GetCounter { get; } = ++a;
  public int ExprCounter => ++b;
}
```

A. 1,1,1,1

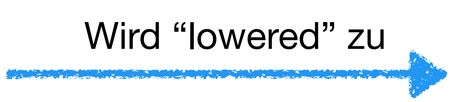
B. 1,2,1,2

C. 1,1,1,2

D. 1,2,1,1

Expression member VS get w/ backing field

```
var entity = new MyEntity();

Console.Write($"{entity.GetCounter}, ");
Console.Write($"{entity.GetCounter}, ");
Console.Write($"{entity.ExprCounter}, ");
Console.Write($"{entity.ExprCounter}");


public class MyEntity
{
    private static int a = 0;
    private static int b = 0;

    public int GetCounter { get; } = ++a;
    public int ExprCounter => ++b;
}
```

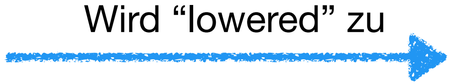



```
public class MyEntity
{
    private static int a;
    private static int b;

    [CompilerGenerated]
    private readonly int <GetCounter>k__BackingField = ++a;

    public int GetCounter
    {
        [CompilerGenerated]
        get
        {
            return <GetCounter>k__BackingField;
        }
    }

    public int ExprCounter
    {
        get
        {
            return ++b;
        }
    }
}
```

- Expression Bodied member getter wird **jedes mal** aufgerufen
- Backing fields werden nur einmal initalisiert

foreach Array

```
var array = new[] { 1, 2 };

foreach(var item in array)
    Console.Write(item);
```




```
int[] array = new int[2];
array[0] = 1;
array[1] = 2;
int[] array2 = array;
int num = 0;
while (num < array2.Length)
{
    int value = array2[num];
    Console.Write(value);
    num++;
}
```

- Kein Collection initializer
- Kein **foreach** mehr im lowered Code
  - Wird in eine **while** loop übersetzt
  - Auch **for** loops werden in eine **while** loop übersetzt

foreach List

```
var list = new List<int> { 1, 2 };

foreach(var item in list)
    Console.Write(item);
```

Wird “lowered” zu

```
List<int> list = new List<int>();
list.Add(1);
list.Add(2);
List<int>.Enumerator enumerator =
    list.GetEnumerator();
try
{
    while (enumerator.MoveNext())
    {
        Console.Write(enumerator.Current);
    }
}
finally
{
    ((IDisposable)enumerator).Dispose();
}
```

- Immer noch kein **foreach**
- Aber wir haben “**Enumeratoren**”
- Und einen **try-finally** block

using in Kombination mit async/await

```
Task<string> GetContentFromUrlAsync(string url)
{
    // Don't do this! Creating new HttpClients
    // is expensive and has other caveats
    // This is for the sake of demonstration
    using var client = new HttpClient();
    return client.GetStringAsync(url);
}
```

using in Kombination mit async/await

```
Task<string> GetContentFromUrlAsync(string url)
{
    // Don't do this! Creating new HttpClients
    // is expensive and has other caveats
    // This is for the sake of demonstration
    using var client = new HttpClient();
    return client.GetStringAsync(url);
}
```

using in Kombination mit async/await

```
Task<string> GetContentFromUrlAsync(string url)
{
    // Don't do this! Creating new HttpClients
    // is expensive and has other caveats
    // This is for the sake of demonstration
    using var client = new HttpClient();
    return client.GetStringAsync(url);
}
```

Wird “lowered” zu

```
HttpClient httpClient = new HttpClient();
try
{
    return httpClient.GetStringAsync(url);
}
finally
{
    if (httpClient != null)
    {
        ((IDisposable)httpClient).Dispose();
    }
}
```

- **using** garantiert* das Disposen via **finally** Blocks
- Der finally Block wird **nach** dem **return** statement ausgeführt
- Dies disposed den **HttpClient** und deshalb wird der Wartende unseres Aufrufes begrüsst mit einer **ObjectDisposedException**

* Wenn man nicht den Stecker zieht, von einem Meteor getroffen wird oder das Program per Task-Manager killt

Weglassen von await

```
try
{
    await DoWorkWithoutAwaitAsync();
}
catch (Exception e)
{
    Console.WriteLine(e);
}

static Task DoWorkWithoutAwaitAsync()
    => ThrowExceptionAsync();

static async Task ThrowExceptionAsync()
{
    await Task.Yield();
    throw new Exception("Hey");
}
```

Ausgabe

```
System.Exception: Hey
   at Program.<<Main>$>g__ThrowExceptionAsync|0_1() in
/Users/stgi/repos/Benchmark/Program.cs:line 19
   at Program.<Main>$(String[] args) in
/Users/stgi/repos/Benchmark/Program.cs:line 6
```

- Die nicht **awaited** Funktion (DoWorkWithoutAwaitAsync) ist **nicht** im stack trace

Weglassen von await

```
try
{
    await DoWorkWithoutAwaitAsync();
}
catch (Exception e)
{
    Console.WriteLine(e);
}

static Task DoWorkWithoutAwaitAsync()
    => ThrowExceptionAsync();

static async Task ThrowExceptionAsync()
{
    await Task.Yield();
    throw new Exception("Hey");
}
```

Wird “lowered” zu

```
[System.Runtime.CompilerServices.NullableContext(1)]
[CompilerGenerated]
internal static Task <<Main>$>g__DoWorkWithoutAwaitAsync|0_0()
{
    return <<Main>$>g__ThrowExceptionAsync|0_1();
}

[System.Runtime.CompilerServices.NullableContext(1)]
[AsyncStateMachine(typeof(<<<Main>$>g__ThrowExceptionAsync|0_1>d))]
[CompilerGenerated]
internal static Task <<Main>$>g__ThrowExceptionAsync|0_1()
{
    <<<Main>$>g__ThrowExceptionAsync|0_1>d stateMachine
        = default(<<<Main>$>g__ThrowExceptionAsync|0_1>d);

    stateMachine.<>t__builder = AsyncTaskMethodBuilder.Create();
    stateMachine.<>1__state = -1;
    stateMachine.<>t__builder.Start(ref stateMachine);
    return stateMachine.<>t__builder.Task;
}
```

- Kein **async** -> Keine State-Machine

Weglassen von await

```
try
{
    await DoWorkWithoutAwaitAsync();
}
catch (Exception e)
{
    Console.WriteLine(e);
}

static Task DoWorkWithoutAwaitAsync()
    => ThrowExceptionAsync();

static async Task ThrowExceptionAsync()
{
    await Task.Yield();
    throw new Exception("Hey");
}
```

Wird “lowered” zu

```
stateMachine
```

Weglassen von await

```
try
{
    await DoWorkWithoutAwaitAsync();
}
catch (Exception e)
{
    Console.WriteLine(e);
}

static Task DoWorkWithoutAwaitAsync()
    => ThrowExceptionAsync();

static async Task ThrowExceptionAsync()
{
    await Task.Yield();
    throw new Exception("Hey");
}
```


```
try
{
    YieldAwaitable.YieldAwaiter awaiter;
    // Here is some other stuff
    awaiter.GetResult();
    throw new Exception("Hey");
}
catch (Exception exception)
{
    <>1__state = -2;
    <>t__builder.SetException(exception);
}
```

- Exceptions werden nicht geworfen, sondern als State auf dem **Task** Objekt gehalten 👋Hallo👋 **async void**
- Aber warum ist der Caller da nicht drauf?

“A stack trace does not tell
you where you came from.

A stack trace tells you
where you are going next.” - *Eric Lippert*

Weglassen von await

```
try
{
    await DoWorkWithoutAwaitAsync();
}
catch (Exception e)
{
    Console.WriteLine(e);
}

static Task DoWorkWithoutAwaitAsync()
    => ThrowExceptionAsync();

static async Task ThrowExceptionAsync()
{
    await Task.Yield();
    throw new Exception("Hey");
}
```

Weglassen von await

```
try
{
    await DoWorkWithoutAwaitAsync();
}
catch (Exception e)
{
    Console.WriteLine(e);
}

static Task DoWorkWithoutAwaitAsync()
    => ThrowExceptionAsync();

static async Task ThrowExceptionAsync()
{
    await Task.Yield();
    throw new Exception("Hey");
}
```

Weglassen von await

```
try
{
    await DoWorkWithoutAwaitAsync();
}
catch (Exception e)
{
    Console.WriteLine(e);
}

static Task DoWorkWithoutAwaitAsync()
    => ThrowExceptionAsync();

static async Task ThrowExceptionAsync()
{
    await Task.Yield();
    throw new Exception("Hey");
}
```

Weglassen von await

```
try
{
    await DoWorkWithoutAwaitAsync();
}
catch (Exception e)
{
    Console.WriteLine(e);
}

static Task DoWorkWithoutAwaitAsync()
    => ThrowExceptionAsync();

static async Task ThrowExceptionAsync()
{
    await Task.Yield();
    throw new Exception("Hey");
}
```

Weglassen von await

```
try
{
    await DoWorkWithoutAwaitAsync();
}
catch (Exception e)
{
    Console.WriteLine(e);
}

static Task DoWorkWithoutAwaitAsync()
    => ThrowExceptionAsync();

static async Task ThrowExceptionAsync()
{
    await Task.Yield();
    throw new Exception("Hey");
}
```

Weglassen von await

```
try
{
    await DoWorkWithoutAwaitAsync();
}
catch (Exception e)
{
    Console.WriteLine(e);
}

static Task DoWorkWithoutAwaitAsync()
    => ThrowExceptionAsync();

static async Task ThrowExceptionAsync()
{
    await Task.Yield();
    throw new Exception("Hey");
}
```

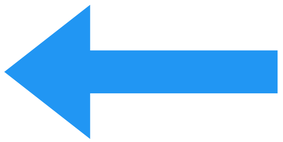
Weglassen von await

```
try
{
    await DoWorkWithoutAwaitAsync();
}
catch (Exception e)
{
    Console.WriteLine(e);
}

static Task DoWorkWithoutAwaitAsync()
    => ThrowExceptionAsync();

static async Task ThrowExceptionAsync()
{
    await Task.Yield();
    throw new Exception("Hey");
}
```

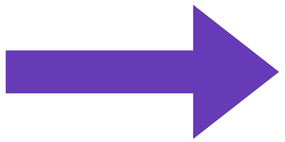
Weglassen von await

```
try
{
    await DoWorkWithoutAwaitAsync();
}
catch (Exception e)
{
    Console.WriteLine(e);
}

static Task DoWorkWithoutAwaitAsync()
    => ThrowExceptionAsync();

static async Task ThrowExceptionAsync()
{
    await Task.Yield();
    throw new Exception("Hey");
}
```

Yield



Collection literals



Yield

```
static IEnumerable<int> GetTenRandomNumbers()
{
    for (var i = 0; i < 10; i++)
        yield return Random.Shared.Next();
}
```

Yield

```
static IEnumerable<int> GetTenRandomNumbers()
{
    for (var i = 0; i < 10; i++)
        yield return Random.Shared.Next();
}
```


```
static List<int> GetTenRandomNumbersList()
{
    var list = new List<int>();
    for (var i = 0; i < 10; i++)
        list.Add(Random.Shared.Next());
    return list;
}
```

Yield

```
static IEnumerable<int> GetTenRandomNumbers()
{
    for (var i = 0; i < 10; i++)
        yield return Random.Shared.Next();
}
```

Yield

```
class GetTenRandomNumbersEnumerable : IEnumerable<int>, IEnumerator<int>
{
    private int state;
    private int current;
    private int i;

    public int Current => current;
    object IEnumerator.Current => Current;
    public void Dispose() { }

    public IEnumerator<int> GetEnumerator() => this;
    IEnumerator IEnumerable.GetEnumerator() => this;

    public bool MoveNext()
    {
        if (state != 0)
        {
            // We were done once and MoveNext was called again
            if (state != 1)
                return false;

            state = -1;
            i++;
        }
        else
        {
            state = -1;
            i = 0;
        }

        // We are done! (false means we don't move to the next element)
        if (i >= 10)
            return false;

        current = Random.Shared.Next();
        state = 1;
        return true;
    }
}
```

Yield

```
class GetTenRandomNumbersEnumerable : IEnumerable<int>, IEnumerator<int>
{
    private int state;
    private int current;
    private int i;

    public int Current => current;
    object IEnumerator.Current => Current;
    public void Dispose() { }

    public IEnumerator<int> GetEnumerator() => this;
    IEnumerator IEnumerable.GetEnumerator() => this;

    public bool MoveNext()
    {
        if (state != 0)
        {
            // We were done once and MoveNext was called again
            if (state != 1)
                return false;

            state = -1;
            i++;
        }
        else
        {
            state = -1;
            i = 0;
        }

        // We are done! (false means we don't move to the next element)
        if (i >= 10)
            return false;

        current = Random.Shared.Next();
        state = 1;
        return true;
    }
}
```

```
Console.WriteLine("Start Here");
foreach(var   item in   GetTenRandomNumbers())
    Console.WriteLine(item);
Console.WriteLine("End Here");
```

state: 0 1

i: 0 1

current: 36 36

Yield

```
[Benchmark]
public int GetSumOf1000NumbersViaYield()
{
    return Get1000RandomNumbers().Sum();
}

[Benchmark]
public int GetSumOf1000NumbersViaList()
{
    return Get1000RandomNumbersList().Sum();
}
```

```
private static IEnumerable<int> Get1000RandomNumbers()
{
    for (var i = 0; i < 1000; i++)
        yield return i;
}

private static List<int> Get1000RandomNumbersList()
{
    var list = new List<int>(1000);
    for (var i = 0; i < 1000; i++)
        list.Add(i);
    return list;
}
```

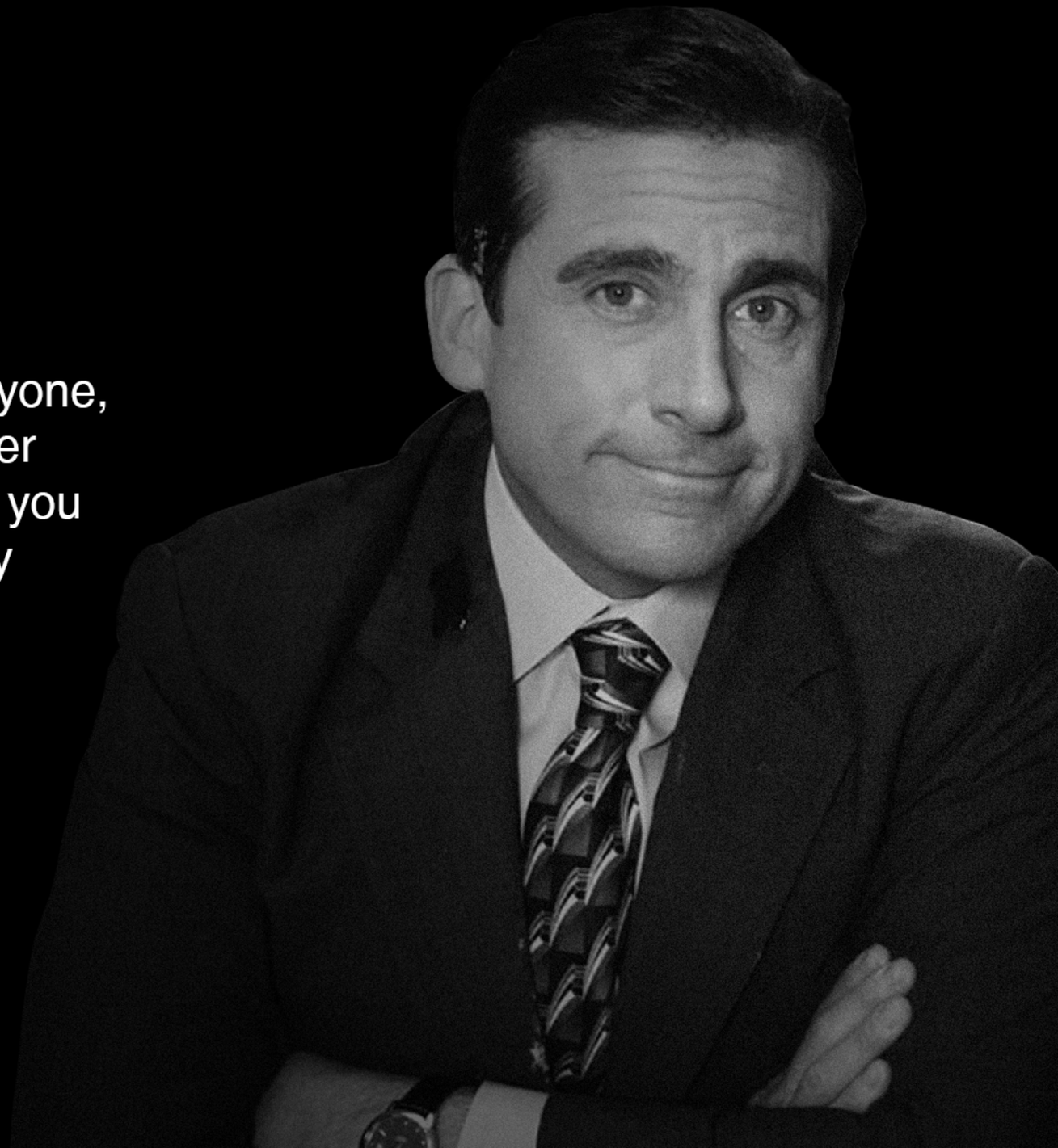
Yield

```
[Benchmark]
public int GetSumOf20NumbersViaYield()
{
    return Get1000RandomNumbers().Take(20).Sum();
}

[Benchmark]
public int GetSumOf20NumbersViaList()
{
    return Get1000RandomNumbersList().Take(20).Sum();
}
```

```
private static IEnumerable<int> Get1000RandomNumbers()
{
    for (var i = 0; i < 1000; i++)
        yield return i;
}

private static List<int> Get1000RandomNumbersList()
{
    var list = new List<int>(1000);
    for (var i = 0; i < 1000; i++)
        list.Add(i);
    return list;
}
```

Don't ever, for any reason, do anything, to anyone, for any reason, ever, no matter what, no matter where, or who, or who you are with, or where you are going, or where you've been, ever, for any reason whatsoever.

- Michael Scott



Das wars!