

# Module 08

## "New Features in C# 6.0"

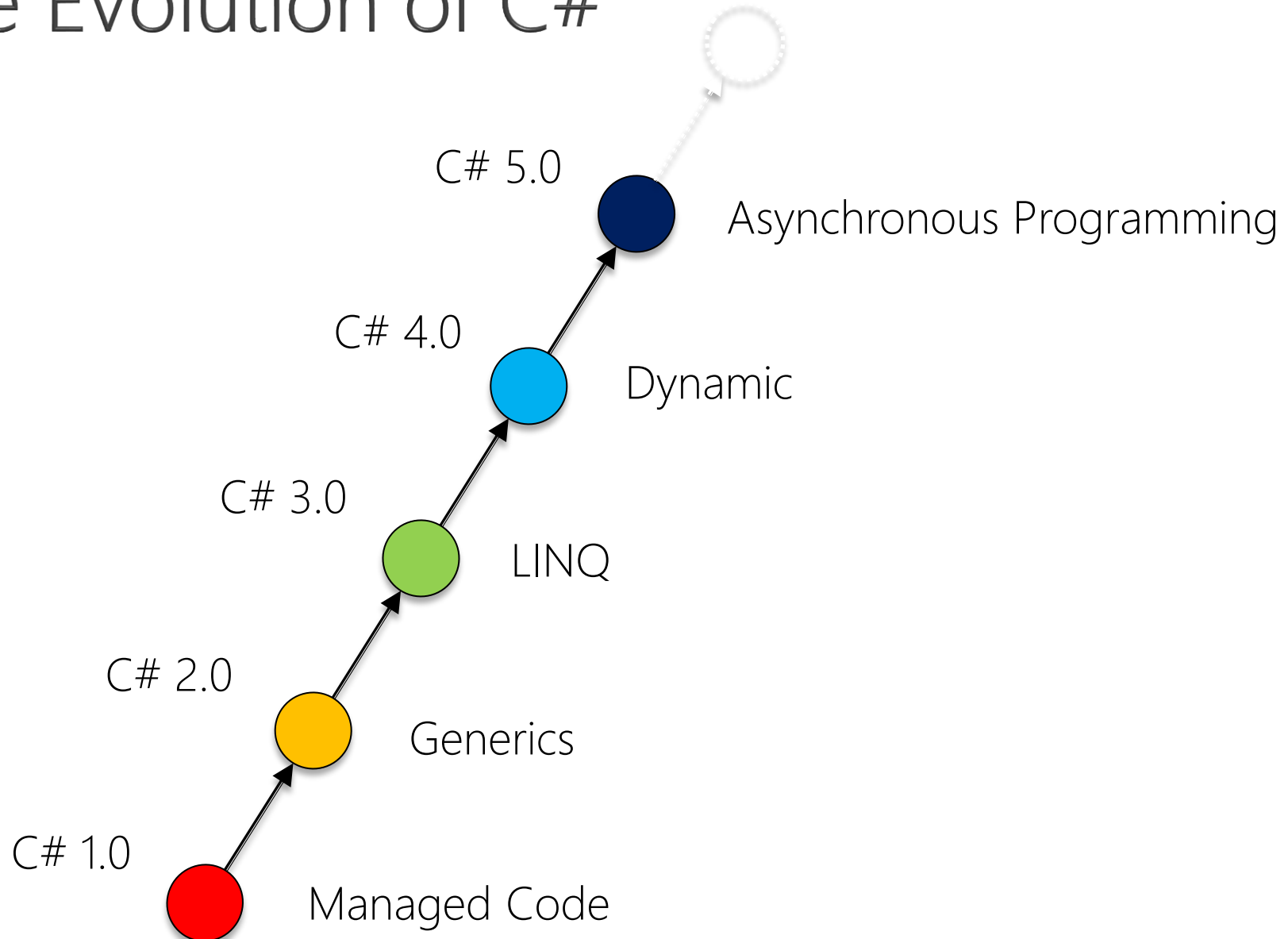


**TEKNOLOGISK**  
**INSTITUT**

# Agenda

- ▶ **Introduction**
- ▶ Namespace-level Features
- ▶ Expression-level Features
- ▶ Statement-level Features
- ▶ Member Declaration Features
- ▶ Scrapped Features for C# 6.0 (*If Time Permits...*)
- ▶ Lab 8
- ▶ Discussion and Review

# The Evolution of C#



# What is new in C# 6.0?

"Less code!" 😊

... but no major additions in terms of languages features

However...

# .NET Compiler Platform (Formerly known as "Roslyn")

- ▶ Project running at Microsoft for more than 4 years
  - Reimplementation of C# and VB.NET compilers
- ▶ Provides an syntax tree representation of programs
- ▶ "Compiler-as-a-Service"
- ▶ At Build 2014 Microsoft announced the open-sourcing of the C# and VB.NET compilers
  - Everyone can "fork" the compiler, but...
- ▶ Contains a really useful Diagnostic Analyzer API

# Agenda

- ▶ Introduction
- ▶ **Namespace-level Features**
- ▶ Expression-level Features
- ▶ Statement-level Features
- ▶ Member Declaration Features
- ▶ Scrapped Features for C# 6.0 (*If Time Permits...*)
- ▶ Lab 8
- ▶ Discussion and Review

# Static Usings

- ▶ Static members can now be imported with **using static**
  - Rules similar to using extension methods

```
using static System.Console;  
  
class Program  
{  
    static void Main(string[] args)  
    {  
        WriteLine( "Hello, World!" );  
    }  
}
```

- ▶ Extension methods are static methods, but intended as instance members

```
using static System.Linq.Enumerable;  
...  
IEnumerable<int> numbers = Range( 0, 100 );    // OK!  
var query = Where( numbers, i => i % 2 == 0 ); // Not OK!
```



# Agenda

- ▶ Introduction
- ▶ Namespace-level Features
- ▶ **Expression-level Features**
- ▶ Statement-level Features
- ▶ Member Declaration Features
- ▶ Scrapped Features for C# 6.0 (*If Time Permits...*)
- ▶ Lab 8
- ▶ Discussion and Review



# Null-conditional Operator

- ▶ C# 6.0 introduces a new null-conditional operator `?.`
  - Also known as the “Elvis operator” 😊
  - Works for reference types and nullable types

```
Person p = ...;  
string s = p?.FullName; // Full name (or null if person == null)
```

- ▶ Right-hand side only evaluated if left-hand side is not null
  - Propagates null through expression
- ▶ Interacts brilliantly with the null-coalescing operator `??`

```
string s = p?.FullName ?? "No name";
```

- ▶ Resolves the well-known design flaw for events
  - Note: `.Invoke()` must be explicitly applied for delegates



# String Interpolation

- ▶ C# 6.0 offers easier formatting of strings

```
public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public string FullName
    {
        get
        {
            return $"{FirstName} {LastName}";
        }
    }
}
```

- ▶ Very flexible
  - Works with virtually any expression
  - Respects usual formatting controls



# nameof Expressions

- ▶ The **nameof** operator provides “reflection-style” access to the name of the syntactical structure

```
public void Add( Person p )  
{  
    if( p == null )  
    {  
        throw new ArgumentNullException( nameof( p ) );  
    }  
    _persons.Add(p);  
}
```

- ▶ Works with more elaborate dotted names

```
Console.WriteLine( nameof( Person.FullName ) );
```

- ▶ Continues the trend of Caller Info Attributes of C# 5.0
  - Eases implementation of **INotifyPropertyChanged** etc.



# Index Initializers

- ▶ Index initializers are now provided for collection initializers

```
var lineUp = new Dictionary<int, Person>
{
    [19] = new Person { FirstName = "Kim", LastName = "Aabech" },
    [11] = new Person { FirstName = "Jesper", LastName = "Lange" },
    [14] = new Person { FirstName = "Ahmed", LastName = "Yasin" }
};
```

- ▶ Intended for scenarios such as **JsonObject** initialization

```
var json = new JsonObject {
    ["FirstName"] = FirstName,
    ["LastName"] = LastName,
    ["Age"] = Age
};
```

- ▶ Note:
  - {...,...} maps to the **Add()** method
  - [...] = ... maps to the underlying indexer



# Add() Extension Methods

- Collection initializer syntax is now extensible via extension methods

```
public static class QueueExtensions
{
    public static void Add<T>(this Queue<T> queue, T t)
    {
        queue.Enqueue(t);
    }
}
```

```
var persons = new Queue<Person>
{
    new Person { FirstName = "Kim", LastName = "Aabech" },
    new Person { FirstName = "Jesper", LastName = "Lange" },
    new Person { FirstName = "Ahmed", LastName = "Yasin" }
};
```



# Agenda

- ▶ Introduction
- ▶ Namespace-level Features
- ▶ Expression-level Features
- ▶ **Statement-level Features**
- ▶ Member Declaration Features
- ▶ Scrapped Features for C# 6.0 (*If Time Permits...*)
- ▶ Lab 8
- ▶ Discussion and Review

# Exception Filters

- ▶ Exception filters facilitates the handling of exceptions matching a specific type and/or predicate

```
var from = Bank.CreateAccount(100);  
var to = Bank.CreateAccount(100);  
  
try  
{  
    Bank.TransferFunds(from, 200, to);  
}  
catch (InsufficientFundsException e) when (e.Account?.IsVIP == true)  
{  
    // Handle VIP account  
}
```

- ▶ Distinct clauses can match same exception type but with different conditions



# await in catch and finally

- ▶ C# 6.0 now allows **await** inside
  - catch
  - finally

```
try
{
    Bank.TransferFunds(from, 200, to);
}
catch (InsufficientFundsException e) when (e.Account?.IsVIP == true)
{
    // Handle VIP account
    await logger.LogExceptionAsync(e);
}
finally
{
    await logger.LogAsync("Completed processing of accounts");
}
```





# Agenda

- ▶ Introduction
- ▶ Namespace-level Features
- ▶ Expression-level Features
- ▶ Statement-level Features
- ▶ **Member Declaration Features**
- ▶ Scrapped Features for C# 6.0 (*If Time Permits...*)
- ▶ Lab 8
- ▶ Discussion and Review

# Auto-property Initializers

- ▶ Initializers can now be supplied for the automatic properties introduced in C# 3.0

```
public class Person
{
    public string FirstName { get; set; } = "Jens";
    public string LastName { get; set; } = "Jensen";
    public int Age { get; set; } = 0;
}
```

- ▶ Equivalent to initializers on fields
  - ...but doesn't use setter!
  - Run before constructors
  - Cannot refer to **this** because happen before object is fully constructed
- ▶ Note: Structs still cannot have initializers!



# Getter-only Auto-properties

- ▶ Moreover, the automatic properties can now be getter- or setter-only

```
public class StockChangedEventArgs : EventArgs
{
    public string Ticker { get; }
    public double StockValue { get; }
    public DateTime Timestamp { get; } = DateTime.Now;
}
```

- ▶ This is a really important mechanism for putting mutable and immutable data types on equal terms!
- ▶ Underlying field is created as **readonly**
  - Can still be assigned from the constructor
  - ... but elsewhere not!
- ▶ Note: Can still have initializers!



# Expression-bodied Members

- ▶ All functionality-based members can have expression bodies
  - Methods
  - Properties
  - Indexers

```
public class Person
{
    public string FirstName { get; set; } = "Jens";
    public string LastName { get; set; } = "Jensen";
    public int Age { get; set; } = 0;

    public string FullName => $"{FirstName} {LastName}";

    public override string ToString() =>
        $"{Name} is {Age} year{(Age == 1 ? "" : "s")} old";
}
```



# Agenda

- ▶ Introduction
- ▶ Namespace-level Features
- ▶ Expression-level Features
- ▶ Statement-level Features
- ▶ Member Declaration Features
- ▶ **Scrapped Features for C# 6.0 (*If Time Permits...*)**
- ▶ Lab 8
- ▶ Discussion and Review

# Scrapped Features for C# 6.0

- ▶ Beware! Many articles, videos, blog posts etc. are no longer current...!
  
- ▶ Contrary to what you might read, the following features have all been scrapped from C# 6.0
  - Parameterless Constructors for Structs
  - Primary Constructors
  - Declaration Expressions
  - **var** return value (and **out var**)
  - **IEnumerable** with **params**
  - Event Initializers
  - Binary Literals and Numeric Separators
  - ...

# Scrapped:

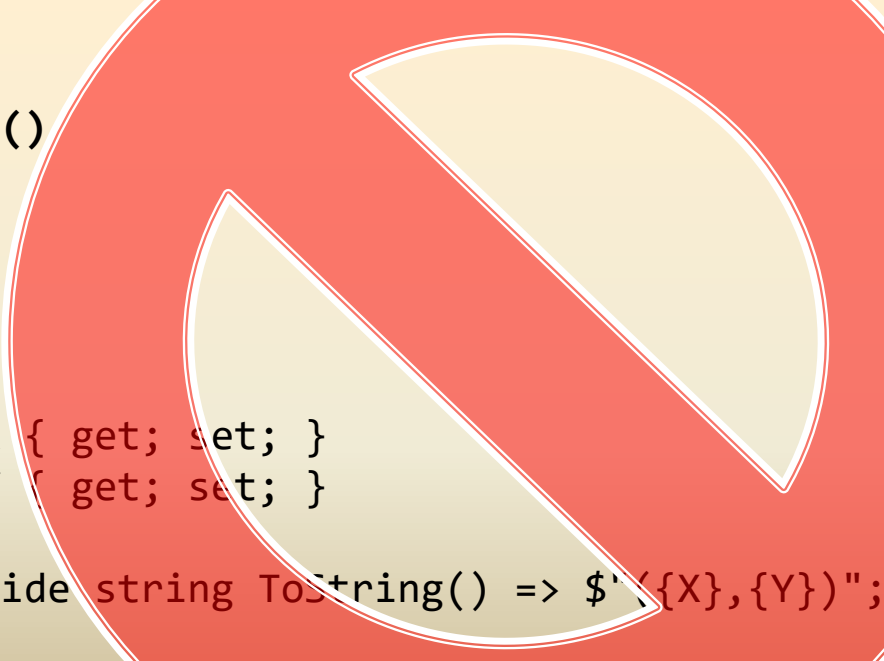
## Parameterless Constructors for Structs

- Until the very(!) last minute we could define parameterless constructors for structs...

```
struct Point
{
    public Point()
    {
        X = -1;
        Y = -1;
    }

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

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



# Scrapped: Primary Constructors

- ▶ Primary constructors were for a very long time hyped as one of the major additions of C# 6.0

```
struct Point( int x, int y )  
{  
    public int X { get; set; } = x;  
    public int Y { get; set; } = y;  
  
    public override string ToString() => $"({X},{Y})";  
}
```

- ▶ But – alas – scrapped!



# Scrapped: Declaration Expressions

- ▶ Allows variable declaration inside of expressions

```
string s = "Yay!";  
if (int.TryParse(s, out var result))  
{  
    Console.WriteLine( result );  
}
```

- ▶ Very many other uses for the construct

# Scrapped: **var** return value and **out var**

- ▶ Automatically inferred types for return values and output parameters

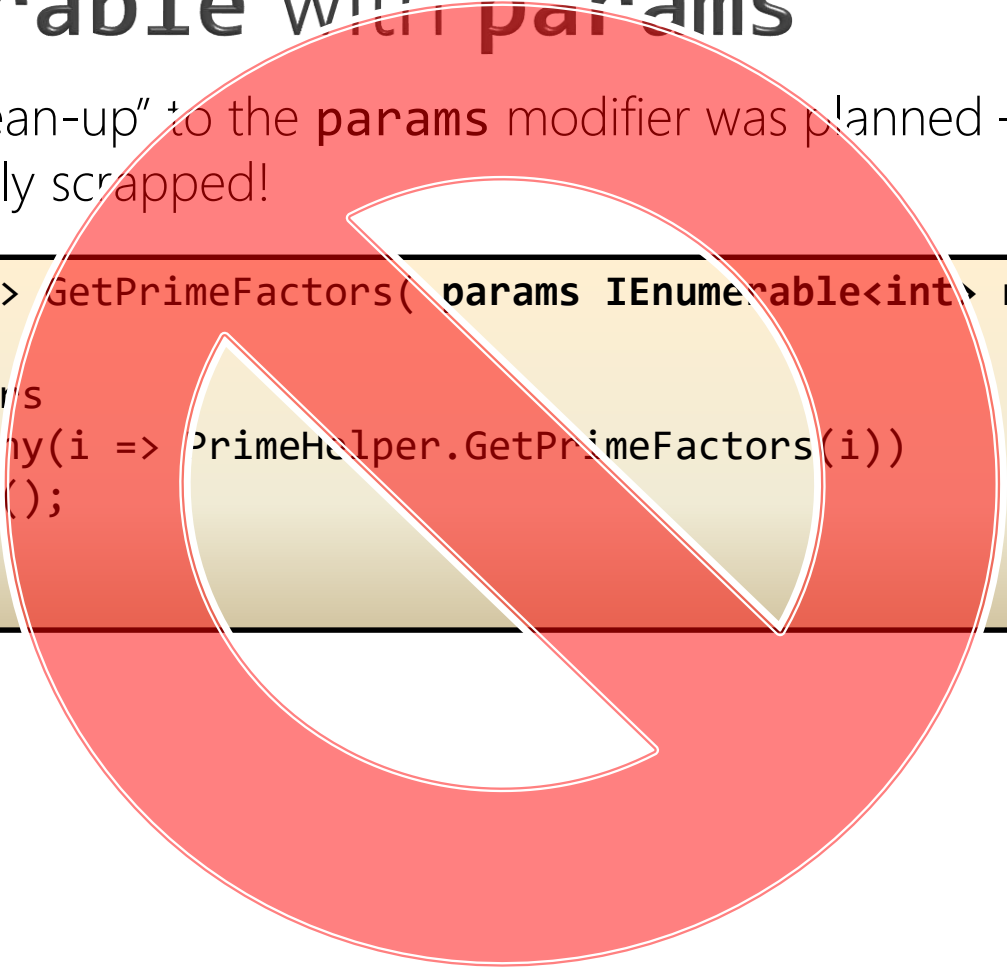
```
var ComputeQuery( int speed, string make )  
{  
    var query = from c in cars  
                 where c.Speed >= speed && c.Make == make  
                 orderby c.Speed  
                 select new { c.Make, c.Color };  
  
    return query;  
}
```

- ▶ Would have been essential to anonymous types!

# Scrapped: **IEnumerable** with **params**

- ▶ A minor “clean-up” to the **params** modifier was planned – but unfortunately scrapped!

```
IEnumerable<int> GetPrimeFactors(params IEnumerable<int> numbers)
{
    return numbers
        .SelectMany(i => PrimeHelper.GetPrimeFactors(i))
        .Distinct();
}
```



# Scrapped: Event Initializers

- ▶ Event initializers for object initializers

```
StockPublisher p1 = new StockPublisher  
{  
    Ticker = "MSFT",  
    StockChanged += ( sender, e ) =>  
        Console.WriteLine( $"{e.Ticker} is priced at {e.StockValue:f2}")  
}
```

# Scrapped: Binary Literals and Numeric Separators

- ▶ Event initializers for object initializers

```
public enum FileAttributes
{
    ReadOnly = 0b00_00_00_00_00_00_01, // 0x0001
    Hidden = 0b00_00_00_00_00_00_10, // 0x0002
    System = 0b00_00_00_00_00_01_00, // 0x0004
    Directory = 0b00_00_00_00_00_10_00, // 0x0010
    Archive = 0b00_00_00_00_01_00_00, // 0x0020
    Device = 0b00_00_00_00_10_00_00, // 0x0040
    Normal = 0b00_00_00_01_00_00_00, // 0x0080
    Temporary = 0b00_00_00_10_00_00_00, // 0x0100
    SparseFile = 0b00_00_01_00_00_00_00, // 0x0200
    ReparsePoint = 0b00_00_10_00_00_00_00, // 0x0400
    Compressed = 0b00_01_00_00_00_00_00, // 0x0800
    Offline = 0b00_10_00_00_00_00_00, // 0x1000
    NotContentIndexed = 0b01_00_00_00_00_00_00, // 0x2000
    Encrypted = 0b10_00_00_00_00_00_00 // 0x4000
}
```

# Lab 8: Write Less Code in C# 6.0

## ► Lab 8.1



# Discussion and Review

- ▶ Introduction
- ▶ Namespace-level Features
- ▶ Expression-level Features
- ▶ Statement-level Features
- ▶ Member Declaration Features
- ▶ Scrapped Features for C# 6.0 (*If Time Permits...*)



WINCUBATE

***Jesper Gulmann Henriksen***

PhD, MCT, MCSD, MCPD

Phone : +45 22 12 36 31

Email : [jgh@wincubate.net](mailto:jgh@wincubate.net)

WWW : <http://www.wincubate.net>

Hasselvangelen 243

8355 Solbjerg

Denmark