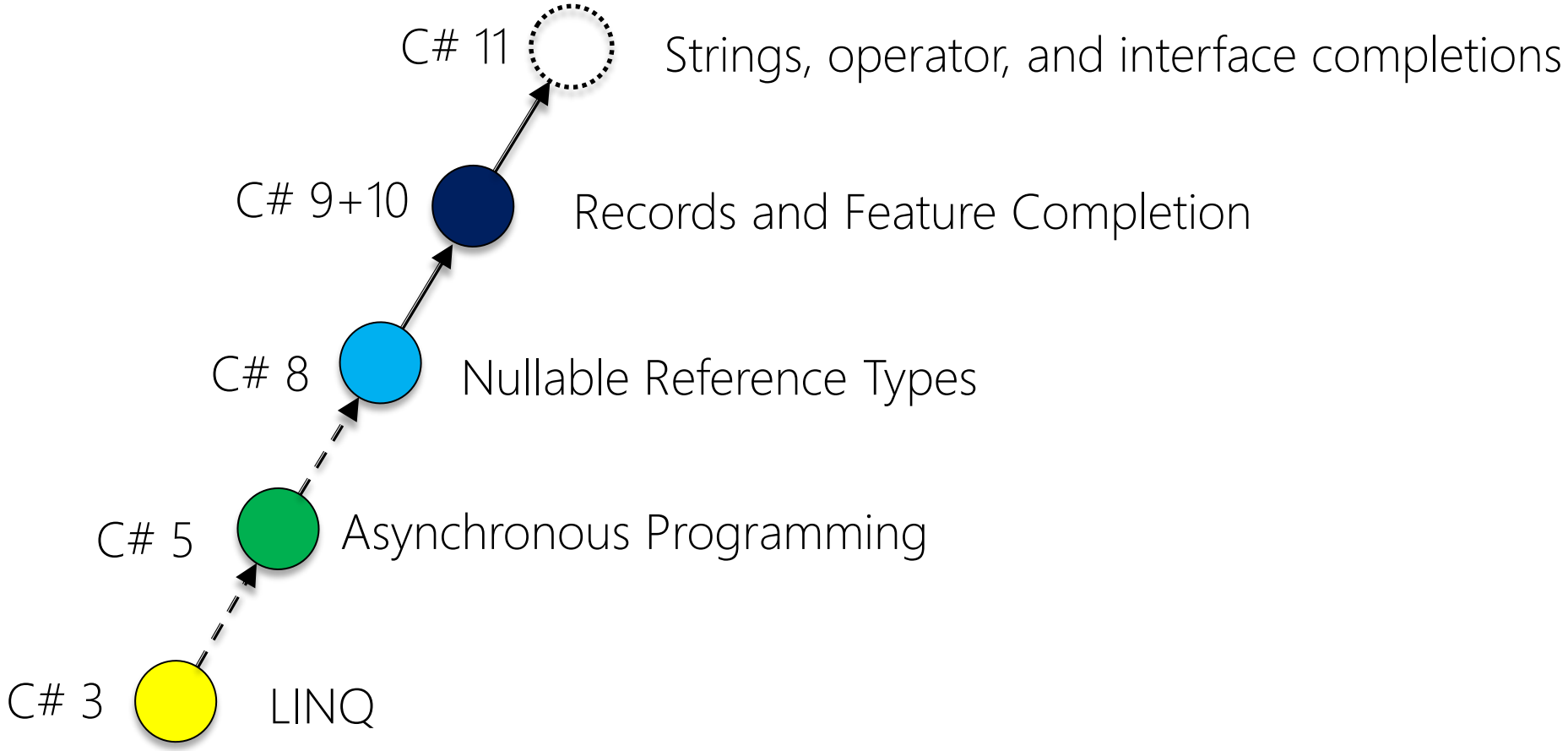
Module 05:

"What's New in C# 11?"

Major Evolutions of C#



Agenda

- Introduction
- **String Improvements**
- Expression Improvements
- Object-Oriented Improvements
- Math and Operators
- Zzzzz…
- Summary

Raw String Literals

- Strings now support multi-line string literals using """

```
string s = """
    Hello,
    "World"
    """;

Console.WriteLine(s);
```

- Excellent for e.g. JSON or XML string literals
- Blocks of *n* "'s in strings can be escaped using *n*+1 "'s in begin and end

- Indentions can also be controlled by ending white-space before """

What about String Interpolation?

- String interpolation proceeds as usual, but might need **$$** and **{{}}** (or more ☺)

```
string firstName = "Jesper";
string lastName = "Gulmann";
string company = "Wincubate ApS";

string s = $$"""
    {
      "firstName": "{{firstName}}",
      "lastName": "{{lastName}}",
      "company": "{{company}}"
    }
    """;
```

- Note: Line breaks are now allowed within string interpolation expressions!

UTF-8 String Literals

```
ReadOnlySpan<byte> s1 = "Hello"u8;

ReadOnlySpan<byte> s2 = """
    Hello,
    "World"
    """u8;
```

- Note:
  - Not strings exactly, but strings already encoded as bytes.
  - Not compile-time constants, because `ReadOnlySpan<byte>` cannot be const

```
var moreBytes = "Hello, "u8 + "World"u8 + "!!"u8;
byte[] moreBytesArray = moreBytes.ToArray();
```

Agenda

- Introduction
- String Improvements
- **Expression Improvements**
- Object-Oriented Improvements
- Math and Operators
- Zzzzz…
- Summary

Pattern-matching Enhancements

- C# 7, 8, 9, and 10 introduced a total of 13 patterns and enhancements

- C# 11 introduces 3 additional list and string patterns or enhancements:
  - List patterns *[a,b,c]* e.g. `[11,22,33]`
  - Slice (or range) patterns .. e.g. `[11, ..]`
  - Spans of chars for constant string "ABC" e.g. `"ABC"`

List Patterns

- Can now match sequences against specific element patterns

```
var elements = new int[] { 11, 22, 33 };

Console.WriteLine(elements is [11, 22, 33]);
Console.WriteLine(elements is [11, 22, 33, 44]);
Console.WriteLine(elements is [>10, <100, 33 or 44]);
```

- Works for types which are *countable* and *indexable*

- Discard pattern _ can be used to match single elements in list patterns

```
Console.WriteLine(elements is [11, _, 33]);
Console.WriteLine(elements is [11, _, _, _]);
```

Slice Patterns

- The Slice (a.k.a. Range) Pattern `..` can be used *at most once* within a list pattern

```
var elements = new int[] { 11, 22, 33 };

Console.WriteLine(elements is [11, ..]);
Console.WriteLine(elements is [.., 33, 44]);
Console.WriteLine(elements is [11, ..] or [.., 44]);
```

- Works for types which are *countable* and *sliceable*

- Slice elements can also be extracted

```
if( elements is [11, ..var sub, _])
{
    // Print sub here
}
```

Character Span Patterns

- Since C# 7 we have been able to match strings on a constant string
- In C# 11 this has been extended to `Span<char>` and `ReadOnlySpan<char>`

```
ReadOnlySpan<char> s1 = "Hello World";
Console.WriteLine(s1 is "Hello");
```

- This way the spans will now work in e.g. switches

```
bool IsKnownAbbreviation(Span<char> s) =>
    s switch
    {
        "etc" or "ie" => true,
        _ => false
    };
```

Extended **nameof** Scope

- The scope of **nameof** has been extended to include
  - Type parameter names
  - Parameter names

```
public static void Validate(
    bool condition,
    [CallerArgumentExpression(nameof(condition))] string? message = null)
{
     if (!condition)
     {
         throw new InvalidOperationException($"Argument failed validation: {message}");
     }
}
```

- Works great for attributes!

Agenda

- Introduction
- String Improvements
- Expression Improvements
- **Object-Oriented Improvements**
- Math and Operators
- Zzzzz…
- Summary

Required Members

- Express that a member must be initialized during construction
  - *Not* required to be initialized to a valid nullable state at the end of the constructor

```
class Person
{
    public required string FirstName { get; init; }
    public string? MiddleName { get; init; }
    public required string LastName { get; init; }
}
```

- Defer the check to the site of object construction

- Help address the shortcoming of nullability checks for reference types of C# 8
- But are actually completely orthogonal to non-nullable reference types
  - Also work for nullable types etc.

[SetsRequiredMembers]

- Asserts that a specific constructor initializes all required members

```
class Person
{
    ...
    [SetsRequiredMembers]
    public Person(string firstName, string lastName)
    {
        FirstName = firstName;
        LastName = lastName;
    }
}
```

- Essentially this is the "**!**" of required members at the constructor level
- Note: Static analysis does *not* check whether correct!

File Accessibility Modifier

- New access modifier on type definitions only
  - Restricts visibility to defining *file*

```
file class C
{
    public static void M()
    {
        Console.WriteLine("Hello from File1");
    }
}
```

- No accessibility modifiers can be used in combination with **`file`**

- Overriding rules apply

Static Abstract Members in Interfaces

- You can add static abstract members in interfaces

```
interface ICanBeEmpty<T>
{
    static abstract T Empty { get; }
}
```

- Can define static abstract properties, methods, events, and operators
  - We will make crucial use of this in the "Math and Operators" section later!

```
class Person : ICanBeEmpty<Person>
{
    public static Person Empty => new Person { ... };

    ...
}
```

Static Virtual Members in Interfaces

- Similarly, static virtual members are now allowed in interfaces

```
interface ICanCreateDefault<T> where T : ICanCreateDefault<T>, new()
{
    static virtual T CreateDefault() => new();
}
```

- Enables polymorphism where the method called depends on the compile-time type rather than the runtime instance type

- Static members are also allowed to be **`sealed`**

Auto-default Structs

- Structs are now default initialized automatically in C# 11 (*if no field initializers*)

```
struct Money
{
    public int Euro { get; set; }
    public int Cents { get; set; }

    public Money()
    {
        Not needed
        Not needed
    }
}
```

Generic Attributes

- C# 11 finally allows custom generic attributes

```
[AttributeUsage(AttributeTargets.All)]
public class DeveloperAttribute<T> : Attribute
{
    public T Info { get; init; }

    public DeveloperAttribute(T info)
    {
        Info = info;
    }
}
```

Agenda

- Introduction
- String Improvements
- Expression Improvements
- Object-Oriented Improvements
- **Math and Operators**
- Zzzzz…
- Summary

Generic Math Support

- Goal: Use mathematical operators in generic types

- `static abstract` / `virtual` members in interfaces
- checked user defined operators
- relaxed shift operators
- unsigned right-shift operator

INumber<T>

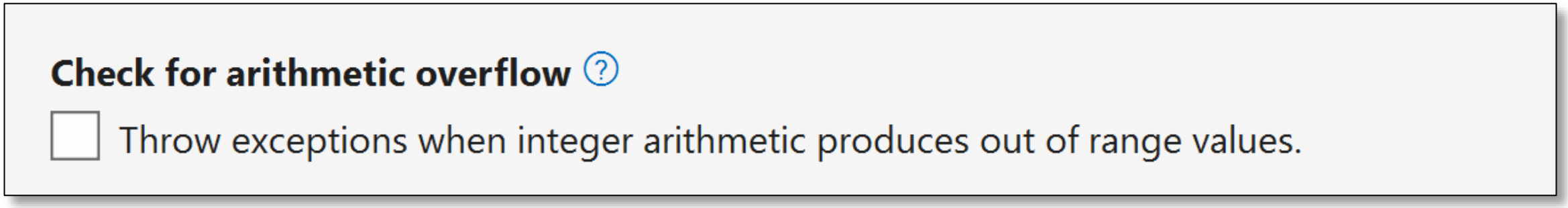
- Math operators are now generic

```
T MultSequence<T>( IEnumerable<T> sequence ) where T : INumber<T>
{
    T total = T.One;
    foreach (T i in sequence)
    {
        total *= i;
    }
    return total;
}
```

Revisiting Checked Contexts

- Since C# 1.0 integral-type aritmetic operations have been performed in either
  - **checked** contexts, or
  - **unchecked** contexts

```
int a = int.MaxValue;
Console.WriteLine(a + 1);
```



User-Defined Checked and Unchecked Operators

- As part of the generic math support, we can define custom checked and unchecked operators

```
record struct Money(int Euro, int Cents)
{
    ...
    public static Money operator +(Money left, Money right) =>
        new(left.TotalCents + right.TotalCents);

    public static Money operator checked +(Money left, Money right)
    {
        checked
        {
            return new(left.TotalCents + right.TotalCents);
        }
    }
}
```

Unsigned Right Shift Operator

- Before C# 11: to force an unsigned right-shift, you would need to
  - cast any signed integer type to an unsigned type
  - perform the shift
  - cast the result back to a signed type

- C# 11 introduces the new **>>>** called *unsigned right shift operator*

```
int x = -8;
int y = x >> 2;
int z = x >>> 2;
```

Relaxing Shift Operator Requirements

- Before C# 11: constraint for x `<<` y or x `>>` y was
  - y must be an integer, or
  - y must be implicitly convertible to an integer

- C# 11 relaxes this constraint to allow the second operand to implementing generic type
  - Or indeed any type ☺

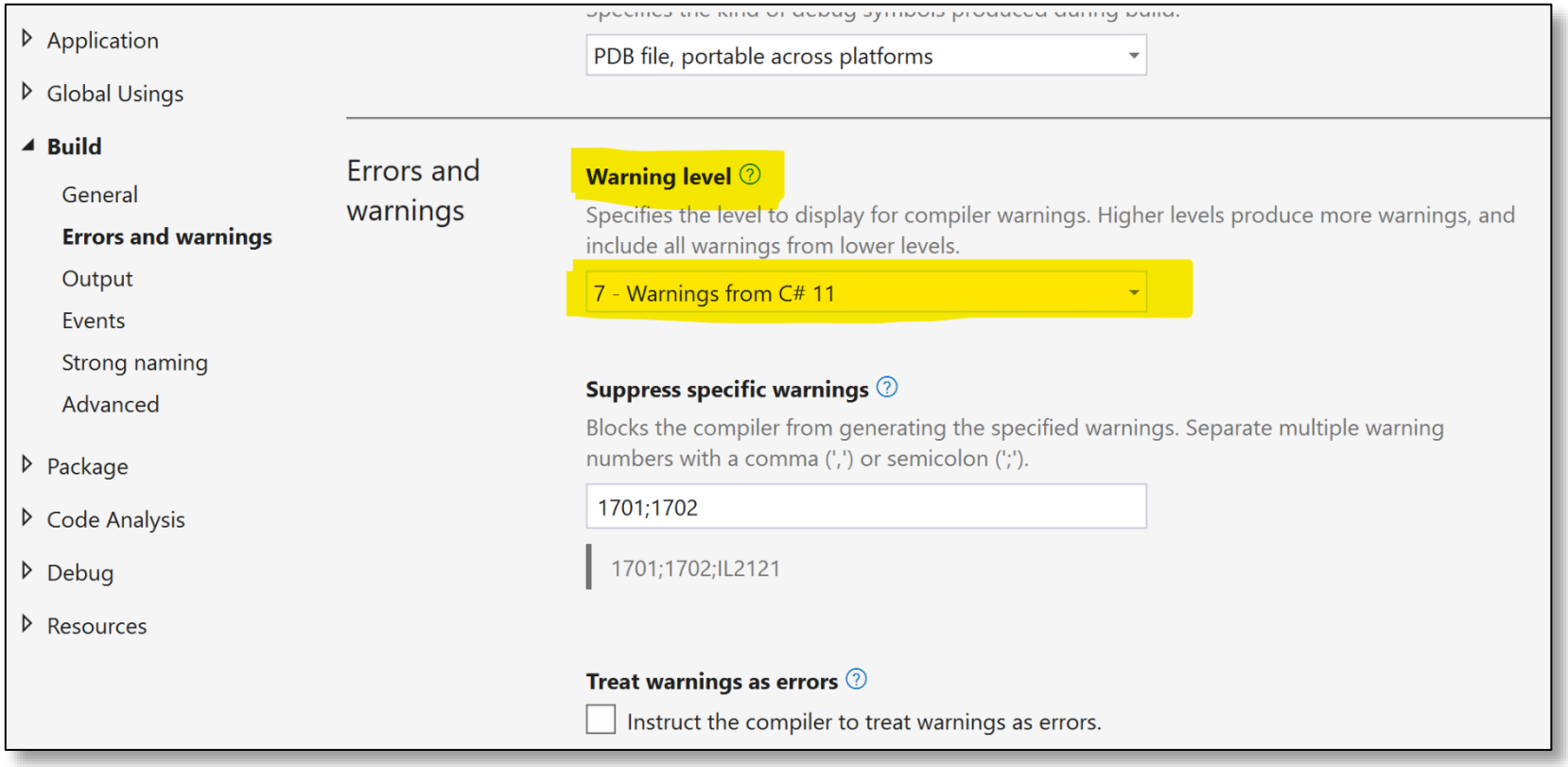
```
record struct Money(int Euro, int Cents)
{
    ...
    public static Money operator <<(Money left, string right) =>
        new(left.TotalCents << right.Length);
}
```

Agenda

- Introduction
- String Improvements
- Expression Improvements
- Object-Oriented Improvements
- Math and Operators
- Zzzzz…
- Summary

C# Warning Waves

- Wave 5 ~ C# 9
- Wave 6 ~ C# 10
- Wave 7 ~ C# 11



Wave 7

- Wave 7 consists of just one additional rule:

- *Any new keywords added for C# will be all lower-case ASCII characters. This warning ensures that none of your types conflict with future keywords. The following code produces CS8981:*

```
class strangelynamedclass
{
    public int X { get; set; }
}
```

`ref` fields and `scoped ref` variables

- `ref struct` can now have `ref` fields

```
readonly ref struct Span<T>
{
    readonly ref T _field;
    readonly int _length;

    public Span(scoped ref T value) { ... }
}
```

- `scoped ref` limits the scope of ref values to e.g. current method

Numeric IntPtr

- `nint` is now type alias for `System.IntPtr`
- `nuint` is now type alias for `System.UIntPtr`

Summary

- Introduction
- String Improvements
- Expression Improvements
- Object-Oriented Improvements
- Math and Operators
- Zzzzz…


WINCUBATE
Jesper Gulmann Henriksen
PhD, MCT, MCSD, MCPD
Phone : +45 22 12 36 31
Email : jgh@wincubate.net
WWW : http://www.wincubate.net
Ringgårdsvej 4A
8270 Højbjerg
Denmark