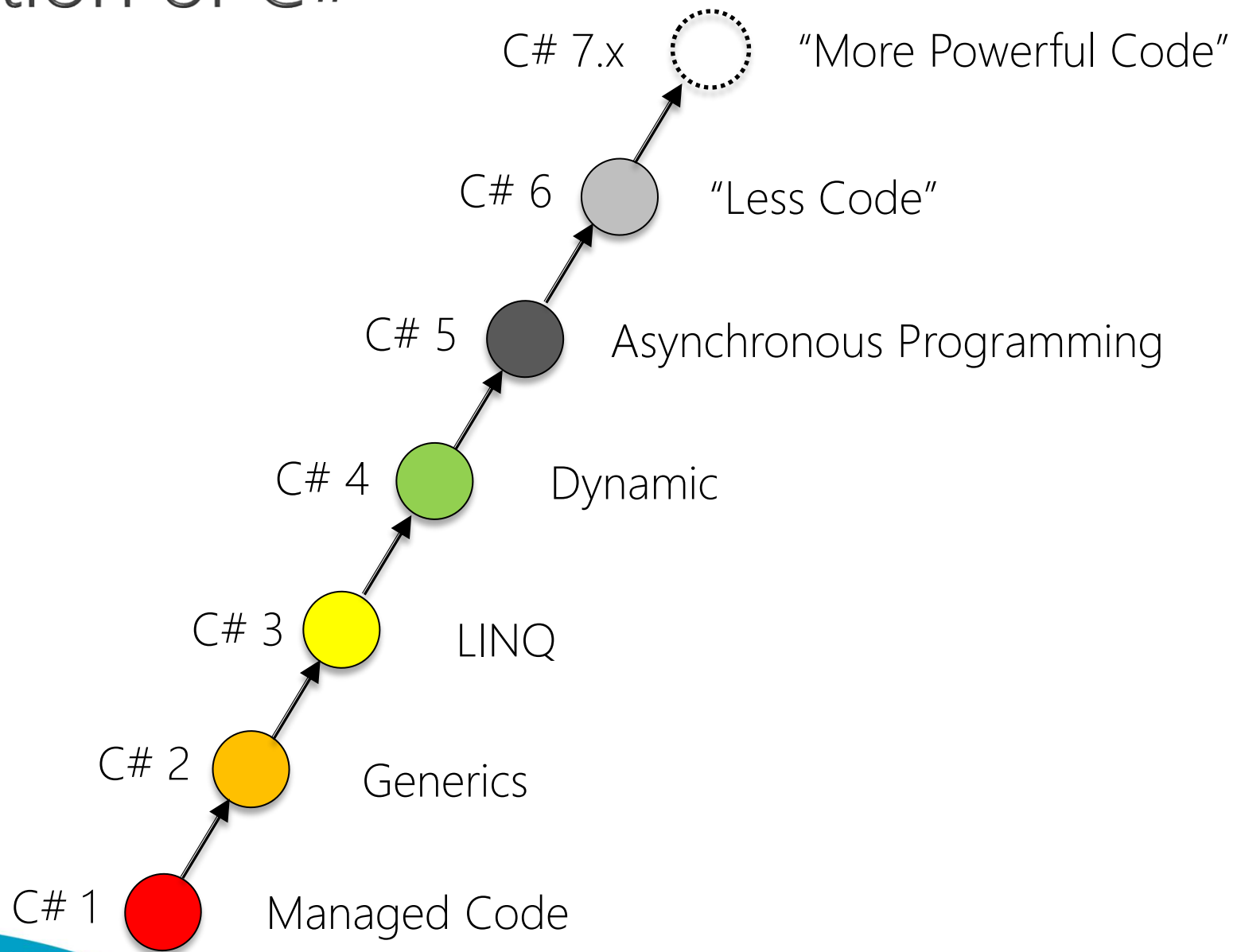


# Module 01:

## "An Introduction to C# 7"



# Evolution of C#



# Agenda

- ▶ Introduction
- ▶ **Value Tuples and Syntax**
- ▶ Pattern Matching
- ▶ Method Improvements
- ▶ Expression Improvements



# Introducing Tuples

- ▶ Not the **Tuple<T1,T2>** type already in .NET 4.0
  - Instead it is a value type with dedicated syntax

```
(int, int) FindVowels( string s )  
{  
    int v = 0;  
    int c = 0;  
    foreach (char letter in s)  
    {  
        ...  
    }  
    return (v, c);  
}
```

```
string input = ReadLine();
```

```
var t = FindVowels(input);  
WriteLine($"There are {t.Item1} vowels and  
{t.Item2} consonants in \"{input}\"");
```

- ▶ Note

- In .NET 4.6.\* projects you must manually add reference to the **System.ValueTuple** nuget package

# Tuple Syntax, Literals, and Conversions

- ▶ Can be easily converted / deconstructed to other names

```
var (vowels, cons) = FindVowels(input);  
(int vowels, int cons) = FindVowels(input);  
  
WriteLine($"There are {vowels} vowels and {cons} consonants in ... ");
```

```
(int vowels, int cons) FindVowels( string s )  
{  
    var tuple = (v: 0, c: 0);  
    ...  
    return tuple;  
}
```

- ▶ Some built-in implicit tuple conversions

- ToString() + Equals() + GetHashCode() (but not == until C# 7.3)



# Custom Tuple Deconstruction

- ▶ Can be easily deconstructed to individual parts

```
(int vowels, int cons) = FindVowels(input);
```

- ▶ Custom types can also be supplied with a *destructor* with out parameters

```
public class Employee
{
    ...
    public void Deconstruct( out string firstName, out string lastName )
    {
        firstName = FirstName;
        lastName = LastName;
    }
}
```

```
Employee elJefe = new Employee { ... };
var (first, last) = elJefe;
WriteLine(first);
```

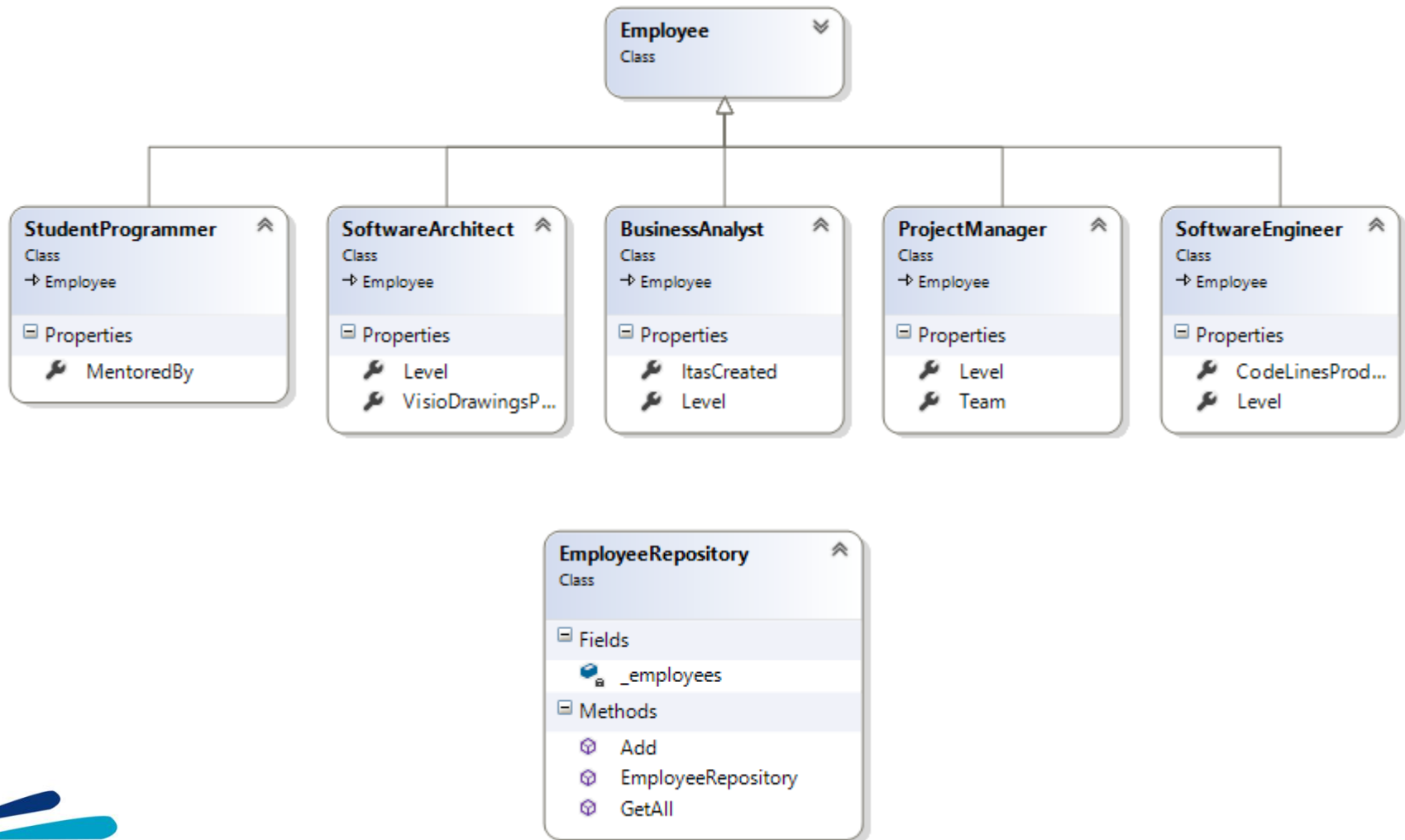
Works for two or more deconstruction parts

# Agenda

- ▶ Introduction
- ▶ Value Tuples and Syntax
- ▶ **Pattern Matching**
- ▶ Method Improvements
- ▶ Expression Improvements



# Example: Employee





# Pattern Matching with `is`

- ▶ Three types of patterns for matching in C# 7
  - Constant patterns      `c`      e.g. `null`
  - Type patterns      `T x`      e.g. `int x`
  - Var patterns      `var x`
- ▶ Matches and/or captures to identifiers to nearest surrounding scope
- ▶ More patterns to come in future C# versions

```
foreach (Employee e in all)
{
    if (e is SoftwareEngineer se)
    {
        WriteLine($"{se.FullName} has produced {se.CodeLinesProduced} " +
                    "lines of C#");
    }
}
```

The `is` keyword is now compatible with patterns



# Type Switch with Pattern Matching

- ▶ Can switch on any type
  - Case clauses can make use of patterns and new **when** conditions

```
Employee e = ...;
switch (e)
{
    case SoftwareArchitect sa:
        WriteLine($"{sa.FullName} plays with Visio");
        break;
    case SoftwareEngineer se when se.Level == SoftwareEngineerLevel.Lead:
        WriteLine($"{se.FullName} is a lead software engineer");
        break;
    case null:
    default:
        break;
}
```

▶ Cases are no longer disjoint – evaluated sequentially!



# Agenda

- ▶ Introduction
- ▶ Value Tuples and Syntax
- ▶ Pattern Matching
- ▶ **Method Improvements**
- ▶ Expression Improvements



# Local Functions

- ▶ Methods within methods can now be defined

```
(int vowels, int cons) FindVowels( string s )  
{  
    ...  
    foreach (char letter in s)  
    {  
        bool IsVowel( char letter )  
        {  
            ...  
        }  
        ...  
    }  
    return tuple;  
}
```

- ▶ Has some advantages
  - Captures local variables
  - Avoids allocations

# Ref Locals

- ▶ Can now create references in the style of C++
  - Similar to the **ref** modifier for parameters

```
int x = 42;  
ref int y = ref x;  
  
x = 87;  
WriteLine(y);
```

- ▶ Ref locals are cannot be reassigned (until C# 7.3)



# Ref Returns

- ▶ Methods can now also return references

```
ref int FindMax( int[] numbers )
{
    int indexOfMax = 0;
    for (int i = 1; i < numbers.Length; i++)
    {
        if (numbers[i] > numbers[indexOfMax])
        {
            indexOfMax = i;
        }
    };

    return ref numbers[indexOfMax];
}
```

Can only return references to heap-based values – not locals

# Agenda

- ▶ Introduction
- ▶ Value Tuples and Syntax
- ▶ Pattern Matching
- ▶ Method Improvements
- ▶ **Expression Improvements**



# More Expression-bodied Members

- ▶ Earlier only getters and methods could be expression-bodied

```
public class Person
{
    ...
    public Person( string name ) => Names.Add(_id, name);

    ~Person() => Names.Remove(_id);

    public string Name
    {
        get => Names[_id];
        set => Names[_id] = value;
    }
}
```

- ▶ New in C# 7.0

- Constructors
- Destructors
- Setters



# Throw Expressions

- ▶ In C# 6 one could not easily just throw an exception in an expression-bodied member
- ▶ C# 7 allows **throw** expressions as subexpressions
  - Also outside of expression-bodied members..!

```
public class EmployeeRepository : IEmployeeRepository
{
    private readonly IList<Employee> _employees;
    ...
    public void Add( Employee employee ) =>
        _employees.Add(employee ??
            throw new ArgumentNullException(nameof(employee)));
}
```

- ▶ Note that a **throw** expression does not have an expression type as such...




# Declaration Expressions:

## out var

- ▶ Introduces local variable in nearest surrounding scope
  - Limitation of general declaration expressions which were scrapped for C# 6

```
string s = ReadLine();  
int result;  
if (int.TryParse(s, out result))  
{  
    WriteLine(result);  
}
```

- ▶ VS 2017 has a handy refactoring for this



```
string s = ReadLine();  
if (int.TryParse(s, out int result))  
{  
    WriteLine(result);  
}
```

- ▶ Note: **return var** is still not in C# 7 ☺

# Discards

- ▶ Temporary, dummy variables which are intentionally unused in application code

```
Employee elJefe = new Employee { ... };  
var (first, _) = elJefe;  
WriteLine(first);
```

```
if (int.TryParse(s, out _))  
{  
    // s is a legal int  
}
```

- ▶ Supported scenarios
  - Tuples and object deconstruction
  - Pattern matching
  - Calls to methods with **out** parameters
  - A standalone **\_** (when no **\_** is in scope)

# Binary Literals and Digit Separators

```
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, // 0x0008
    Archive =           0b00_00_00_00_01_00_00, // 0x0010
    Device =            0b00_00_00_00_10_00_00, // 0x0020
    Normal =            0b00_00_00_01_00_00_00, // 0x0040
    Temporary =         0b00_00_00_10_00_00_00, // 0x0080
    SparseFile =        0b00_00_01_00_00_00_00, // 0x0100
    ReparsePoint =      0b00_00_10_00_00_00_00, // 0x0200
    Compressed =        0b00_01_00_00_00_00_00, // 0x0400
    Offline =           0b00_10_00_00_00_00_00, // 0x0800
    NotContentIndexed = 0b01_00_00_00_00_00_00, // 0x1000
    Encrypted =         0b10_00_00_00_00_00_00 // 0x2000
}
```

# Summary

- ▶ Introduction
- ▶ Value Tuples and Syntax
- ▶ Pattern Matching
- ▶ Method Improvements
- ▶ Expression Improvements





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>

Hasselvangel 243

8355 Solbjerg

Denmark