Module 03:

"A Glimpse of C# 8"

WINCUBATE

Agenda

- **Nullable Reference Types**
- Additional Types, Expressions, and Patterns
- Default Interface Implementations
- Better Async Integration
- Extension Everything

Null References: "The Billion-dollar Mistake"

"I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years."

– Tony Hoare 2009

Step One: Expressing Intent

- Could create new types for reference types. Either
  - A reference is not supposed to be null (e.g. **`string!`**)
  - A reference is allowed to be null (e.g. **`string?`**)

- C# reference types today does not make this distinction!

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

    ...
}
```

Step One: Expressing Intent

- Solution
  - A reference is not supposed to be null: **`string`**
  - A reference is welcome to be null: **`string?`**

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

    ...
}
```

- Note: Breaking change!
  - Use a compiler switch to control behavior

Step Two: Enforcing Behavior

- Do compile-time static analysis warning when
  - Setting a nonnullable to null
  - Dereferencing a nullable reference

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

    public Person( string firstName ) => FirstName = firstName;

    int GetLengthOfMiddleName( Person p ) => p.MiddleName.Length;
}
```

Agenda

- Nullable Reference Types
- **Additional Types, Expressions, and Patterns**
- Default Interface Implementations
- Better Async Integration
- Extension Everything

Records

- Immutable, data-like class types

```
class Person (string First, string Last);
```

- Compiler auto-generates value object

```
class Person : IEquatable<Person>
{
    public string First { get; }
    public string Last { get; }

    public Person(string First, string Last) => { ... }
    public void Deconstruct(out string First, out string Last) => { ... }

    public bool Equals(Person other) => { ... }
    public override bool Equals(object obj) => { ... }
    public override int GetHashCode() => { ... }
    ...
}
```

Recursive Data Types

- Allow definitions of "functional-style" expression types

```
abstract class Expr;

class X() : Expr;

class Const(double Value) : Expr;

class Add(Expr Left, Expr Right) : Expr;

class Mult(Expr Left, Expr Right) : Expr;

class Neg(Expr Value) : Expr;
```

Additional Patterns

- Positional and Recursive Patterns

```
Expr Simplify( Expr e ) =>
    switch (e)
    {
        case Mult(Const(0), _): return Const(0);
        case Mult(_, Const(0)): return Const(0);
        case Mult(Const(1), var x): return Simplify(x);
        case Mult(var x, Const(1)): return Simplify(x);
        case Mult(Const(var l), Const(var r)): return Const(l * r);
        case Add(Const(0), var x): return Simplify(x);
        case Add(var x, Const(0)): return Simplify(x);
        case Add(Const(var l), Const(var r)): return Const(l + r);
        case Neg(Const(var k)): return Const(-k);
        default: return e;
    }
```

Revamped Switch Statement

- More "functional"-inspired **`switch`**

```
string Choose( Employee employee ) =>
     employee switch
     {
         SoftwareArchitect sa => $"Hello, Mr. Architect {sa.LastName}",
         SoftwareEngineer se { LastName: "Gulmann" } => "Please code!",
         StudentProgrammer sp { FirstName: var firstName } =>
             $"Please get coffee and donuts, {firstName}",
         _ => "Have a nice day... :-)"
    }
}
```

Agenda

- Nullable Reference Types
- Additional Types, Expressions, and Patterns
- **Default Interface Implementations**
- Better Async Integration
- Extension Everything

Default Interface Members

- Allow better backwards compatibility in interfaces

```
interface ILogger
{
    void Log(LogLevel level, string message);
    void Log(Exception ex) => Log(LogLevel.Error, ex.ToString());
}
```

```
class ConsoleLogger : ILogger
{
    public void Log(LogLevel level, string message) { ... }
}
```

```
class FileLogger : ILogger
{
    public void Log(LogLevel level, string message) { ... }
    public void Log(Exception ex) { ... }
}
```

Agenda

- Nullable Reference Types
- Additional Types, Expressions, and Patterns
- Default Interface Implementations
- **Better Async Integration**
- Extension Everything

Async Streams

- New async enumerable interfaces with corresponding **foreach** syntax

```
public interface IAsyncEnumerable< out T>
{
    IAsyncEnumerator<T> GetAsyncEnumerator();
}
public interface IAsyncEnumerator<out t>
{
    Task<bool> WaitForNextAsync();
    T TryGetNext( out bool success );
}
```

```
IAsyncEnumerable<int> query = ...;
foreach await( int i in query )
{
    Console.WriteLine( i );
}
```

Async Disposable

- New async interface

```
public interface IAsyncDisposable
{
    Task DisposeAsync();
}
```

- Corresponding async await syntax to use with **`IAsyncDisposable`**

```
IAsyncDisposable resource = new ...;
using await (resource)
{
    resource.DoSomething();
    resource.DoSomethingElse();
}
```

Agenda

- Nullable Reference Types
- Additional Types, Expressions, and Patterns
- Default Interface Implementations
- Better Async Integration
- **Extension Everything**

Extension "Everything"

- C# 3 introduced Extension Methods to facilitate LINQ
- C# 8 might introduce
  - Extension Constructors
  - Extension Properties
  - Extension Events
  - …
- Appealing indeed, but might be somewhat quirky due to "static-ness"

Summary

- Nullable Reference Types
- Additional Types, Expressions, and Patterns
- Default Interface Implementations
- Better Async Integration
- Extension Everything

WINCUBATE


WINCUBATE
Jesper Gulmann Henriksen
PhD, MCT, MCSD, MCPD
Phone : +45 22 12 36 31
Email : jgh@wincubate.net
WWW : http://www.wincubate.net
Hasselvangen 243
8355 Solbjerg
Denmark