

The background of the slide features a wide-angle photograph of a natural landscape at sunset. A large, calm lake or river curves through the center-left of the frame. The sky is filled with dramatic, colorful clouds, ranging from deep blues and purples to bright orange and yellow as the sun sets on the horizon. In the foreground, there's a grassy field and a dense line of green trees separating the viewer from the water.

Bringing C# nullability into existing code

Maarten Balliauw

<https://mastodon.online/@maartenballiauw>

Agenda

- Nullable reference types in C#
- Internals of C# nullable reference types
- Annotating your C# code
- Techniques and tools to update your project

A photograph of a wooden dock extending from the bottom left towards the center of the frame, ending in a small pier. The water is calm with gentle ripples. In the background, a dense forest of evergreen trees lines the shore under a sky filled with scattered, dark clouds.

Nullable reference types in C#

Reference types, value types,
and null

What will be the output of this code?

```
string s = GetValue();
Console.WriteLine($"Length of '{s}': {s.Length}");

string GetValue() => null;
```

Unhandled exception. System.NullReferenceException:
Object reference not set to an instance of an object.
at Program.<Main>\$(<String[] args>) in Program.cs:line 2

Add a check for null

```
string s = GetValue();
Console.WriteLine(s != null
    ? $"Length of '{s}': {s.Length}"
    : "String is null.");
string GetValue() => null;
```

How do you know you need a check?

- For reference types, not particularly clear...
- For value types, easy!
`int? bool? long? decimal? DateTime?`
- ? denotes null is possible

Note: value types can't *really* be null:
compiler magic converts to `Nullable<T>`

Clear intent: ? tells you if null is possible

Reference types

```
string s = GetValue();
Console.WriteLine(s != null
    ? $"Length of '{s}': {s.Length}"
    : "String is null.");
```

```
string GetValue() => null;
```

Value types

```
DateTime? s = GetValue();
Console.WriteLine(s.HasValue
    ? $"The date is: {s.Value:0}"
    : "No date was given.");
```

```
DateTime? GetValue() => null;
```

What are nullable reference types (NRT)?

Nullable reference types (NRT)

- Have always been a part of C#: every reference type is nullable
- C#8+ nullable reference types flip the idea:
 - Non-nullable by default
 - Syntax to annotate a reference type as nullable
- What will be the output of this code?

```
string s = GetValue();
Console.WriteLine($"Length of '{s}': {s.Length}");
```

Unhandled exception. System.NullReferenceException:
Object reference not set to an instance of an object.
at Program.<Main>\$(<Main>String[] args) in Program.cs:line 2

Nullable reference types (NRT)

```
string s = GetValue();
Console.WriteLine($"Length of '{s}': {s.Length}");

string? GetValue() => null;
```

- The above code will emit compiler warnings
 - CS8600 - Converting null literal or possible null value to non-nullable type.
 - CS8602 - Dereference of a possibly null reference.
- Your IDE will show them, too

```
Console.WriteLine($"Length of '{s}': {s.Length}");
```

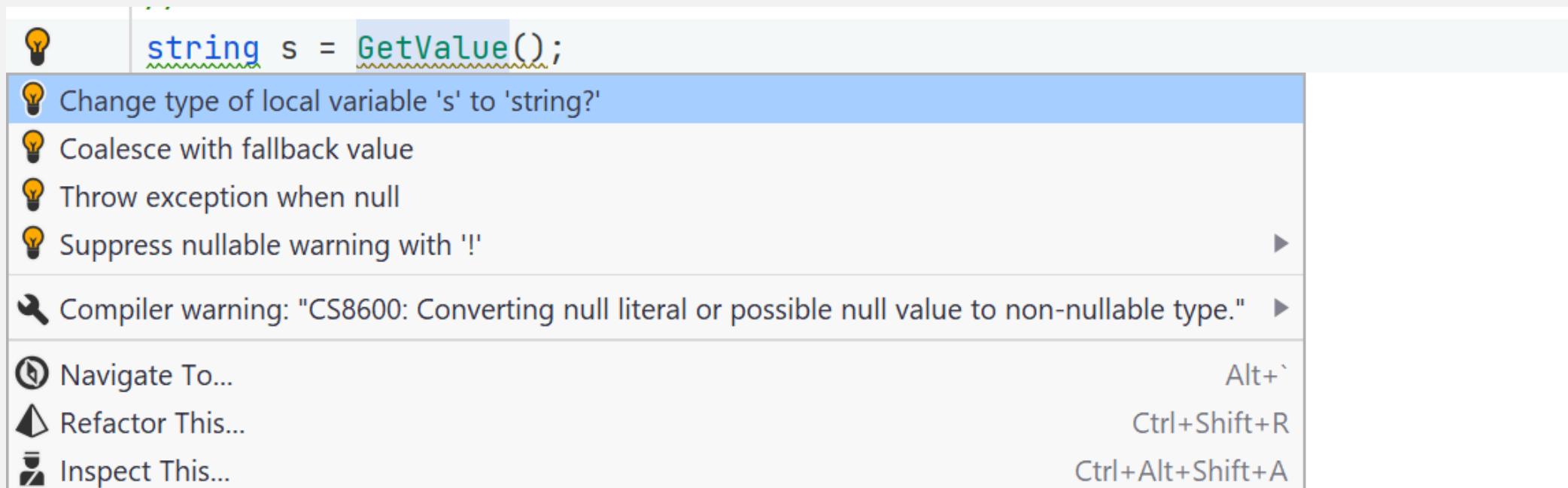
Dereference of a possibly null reference

local variable string s

⋮

Nullable reference types (NRT)

- C#8+ nullable reference types are just *annotations*
- Compiler and IDE help you detect potential null values
- Design-time and compile-time



Flow analysis



Flow analysis

Demo

Flow analysis

- IDE and compiler analyze code flow
 - Code path analysis determines warning/no warning
- Using var is always considered nullable
 - Until flow analysis determines otherwise
- The null-coalescing / null-forgiving operator (“dammit”)
 - Suppress warnings, postfix expression with !
 - May help with directing flow analysis (e.g. while migrating to NRT)...
 - ...but should be considered an **antipattern**.

Summary: Nullable reference types in C#

- No runtime safety
- Design-time and compile-time help to determine if a variable may be null before dereferencing it
- Give better static flow analysis on your code
 - Your null checks will help flow analysis
 - Null-forgiving operator may help if really needed
 - But it is an antipattern!

A close-up view of the red-colored envelope of a hot air balloon. The mesh fabric is visible, and bright orange and yellow flames are visible through the mesh, indicating the balloon is being heated up.

Under the hood

00-BZG

Reference types vs. value types

- Value type: compiler magic to `Nullable<T>`
- Reference type: always nullable
- How do compiler and IDE know about reference type nullability?

Pre-C#8 code

```
#nullable disable
string GetString1() => "";

.method private hidebysig instance string
    GetString1() cil managed
{
    .maxstack 8

    IL_0000: ldstr      ""
    IL_0005: ret
}
```

Pre-C#8 code

```
#nullable disable
string GetString1() => "";

.method private hidebysig instance string
    GetString1() cil managed
{
    .maxstack 8

    IL_0000: ldstr      ""
    IL_0005: ret
}
```

C#8+

```
#nullable enable
string? GetString2() => "";

.method private hidebysig instance string
    GetString2() cil managed
{
    .custom instance void
System.Runtime.CompilerServices.NullableContextAttribute::.ctor([in] unsigned int8)
    = (01 00 02 00 00 ) // ....
    // unsigned int8(2) // 0x02
    .maxstack 8

    IL_0000: ldstr      ""
    IL_0005: ret
}
```

C#8+

```
#nullable enable
string? GetString2() => "";

.method private hidebysig instance string
    GetString2() cil managed
{
    .custom instance void
System.Runtime.CompilerServices.NullableContextAttribute::.ctor([in] unsigned int8)
    = (01 00 02 00 00 ) // ....
    // unsigned int8(2) // 0x02
    .maxstack 8

    IL_0000: ldstr      ""
    IL_0005: ret
}
```

Information for flow analysis

```
.custom instance void
System.Runtime.CompilerServices.NullableContextAttribute::ctor([in] unsigned int8)
    = (01 00 02 00 00) // ....
    // unsigned int8(2) // 0x02
```

- Oblivious: 0
 - Default, pre-C#8 behaviour (everything is maybe null)
- Not annotated: 1
 - Every reference type is non-nullable by default
- Annotated: 2
 - Every reference type has an implicit ?

More attributes!

```
#nullable enable
public string GetString3(string? a, string b, string? c) => "";

.method public hidebysig instance string GetString3(string a, string b, string c)
cil managed
{
    .custom instance void NullableContextAttribute::.ctor([in] unsigned int8)
        = (01 00 01 00 00 ) // int8(1) - by default treat ref. types as not annotated
    .param [1]
        .custom instance void NullableAttribute::.ctor([in] unsigned int8)
            = (01 00 02 00 00 ) // int8(2) - treat param[1] as annotated
    .param [3]
        .custom instance void NullableAttribute::.ctor([in] unsigned int8)
            = (01 00 02 00 00 ) // int8(2) - treat param[3] as annotated
    .maxstack 8
```

More attributes!

```
#nullable enable
public string GetString3(string? a, string b, string? c) => "";

.method public hidebysig instance string GetString3(string a, string b, string c)
cil managed
{
    .custom instance void NullableContextAttribute::.ctor([in] unsigned int8)
        = (01 00 01 00 00) // int8(1) - by default treat ref. types as not annotated
    .param [1]
        .custom instance void NullableAttribute::.ctor([in] unsigned int8)
            = (01 00 02 00 00) // int8(2) - treat param[1] as annotated
    .param [3]
        .custom instance void NullableAttribute::.ctor([in] unsigned int8)
            = (01 00 02 00 00) // int8(2) - treat param[3] as annotated
    .maxstack 8
```

More attributes!

```
#nullable enable
public string GetString3(string? a, string b, string? c) => "";

.method public hidebysig instance string GetString3(string a, string b, string c)
cil managed
{
    .custom instance void NullableContextAttribute::.ctor([in] unsigned int8)
        = (01 00 01 00 00) // int8(1) - by default treat ref. types as not annotated
    .param [1]
        .custom instance void NullableAttribute::.ctor([in] unsigned int8)
            = (01 00 02 00 00) // int8(2) - treat param[1] as annotated
    .param [3]
        .custom instance void NullableAttribute::.ctor([in] unsigned int8)
            = (01 00 02 00 00) // int8(2) - treat param[3] as annotated
    .maxstack 8
```

Information for flow analysis

- Default for method: `NullableContextAttribute`
- Per-parameter overrides: `NullableAttribute`
- C# compiler tries to emit as few attributes as possible
 - Default that applies to most parameters
 - Overrides to that default where needed
 - Reduce overhead when analyzing code during compilation or in IDE

How do you want to surface NRT?

- Nullable *annotation* context
- In code, using `#nullable value`
- In project file `<Nullable>value</Nullable>`
- Possible values:
 - `disable` - pre-C# 8.0 behavior (no NRT)
 - `enable` - all analysis and language features
 - `warnings` - all analysis, and warnings when code might dereference null
 - `annotations` - no analysis, but lets you use ?

Which annotation context to use?

- New projects: enable it.
- Migration: “It depends” 
 - Disable as the default, enable file-per-file until done
 - Enable as the default, live with many warnings as you go through
 - Warnings as the default, enable file-per-file until done
 - See warnings where you can improve
 - Annotations as the default
 - Allow adding ?, but no real benefit.

Summary: Nullable reference types in C#

- Nullable context: what the compiler emits and consumes
- Nullable annotation context: what you want to surface
 - enable by default for new projects
 - disable by default for existing, gradual `#nullable enable`

Annotating your C# code



Is ? enough...

```
#nullable enable

public static string? Slugify(string? value)
{
    if (value == null)
    {
        return null;
    }

    return value.Replace(" ", "-").ToLowerInvariant();
}
```

```
var slug :string? = Slugify("this is fine");
Console.WriteLine(slug.Length);
```

Dereference of a possibly null reference

Is ? enough...

```
public static string? Slugify(string? value)
```

- Returns a non-null string when a non-null parameter is passed
- Returns a null string when a null parameter is passed

Fine-grained annotations!

```
[return: NotNullIfNotNull("value")]
public static string? Slugify(string? value)
```

```
var slug1:string? = Slugify(null);
var slug2:string? = Slugify("this is fine");

Console.WriteLine(slug1.Length);
Console.WriteLine(slug2.Length);
```

Fine-grained annotations!

```
public static bool IsNullOrEmpty(
    [NotNullWhen(false)] string? value)
{
    return value == null || 0 == value.Length;
}
```

```
var s :string? = DateTime.Now.Day == 1 ? "" : null;

Console.WriteLine(s.Length);

if (string.IsNullOrEmpty(s)) return;

Console.WriteLine(s.Length);
```

Fine-grained annotations!

- Preconditions
 - When writing to a parameter, field, or property setter
- Postconditions
 - When reading from a field, property or return value
- Conditional postconditions
 - Make arguments dependent on return value
- Failure conditions
 - Further analysis is not needed

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/attributes/nullable-analysis>



Annotations

Demo

JetBrains.Annotations

- Many libraries ship them, can help with migration
 - Entity Framework Core, Unity, Hangfire, ...
- More powerful, but only in ReSharper and JetBrains Rider
 - [CanBeNull], [NotNull]
 - [ItemNotNull], [ItemCanBeNull]
 - [ContractAnnotation]

```
[ContractAnnotation("json:null => false, person:null")]
public bool TryDeserialize(string? json, out Person? person)
{
    // ...
}
```

https://www.jetbrains.com/help/reSharper/Reference_Code_Annotation_Attributes.html

Nullability and generics

- Use ? where needed

```
List<string>
```

```
List<string?>
```

```
List<string?>?
```

- Use annotations

```
[return: MaybeNull]
```

```
public T Find<T>(IEnumerable<T> sequence, Func<T, bool> predicate)
```

- Use notnull constraint

```
public static void WriteToConsole<T>(T item) where T : notnull
```

```
WriteToConsole(null); // CS8714 doesn't match 'notnull' constraint
```

Referenced code, libraries & frameworks

- Using NRT is easy if your dependencies use it
 - .NET BCL is annotated
 - Many OSS libraries are annotated
- Annotate your own code for dependents
 - Provide design-time and compile-time hints
- Remember NRT are not runtime safety!
 - Consider null checks for code that others consume
 - They may not use C#, or have #nullable disable

Summary: Annotating your C# code

- There are more annotations than ?
- Clearly communicates intent with flow analyzer
- Compiler annotations and JetBrains.Annotations
- You may still need null checks for your libraries

A blurred photograph of a park scene. In the foreground, there's a body of water with ripples and reflections of the surrounding green trees. The background shows more trees and a faint building on the left side.

Techniques and tools
to update your C# project

Default nullable annotation context

- `#nullable enable/disable/warnings/annotations`
- New projects: enable it
- Existing projects:
 - Small: enable it and plow through
 - Large:
 - Enable file by file
 - Combine with warnings at project level

Start at the center and work outwards

- Classes with few or zero dependencies on others
 - Often: DTOs / POCOs
- Typically used by many dependents
 - Small change flows to many places

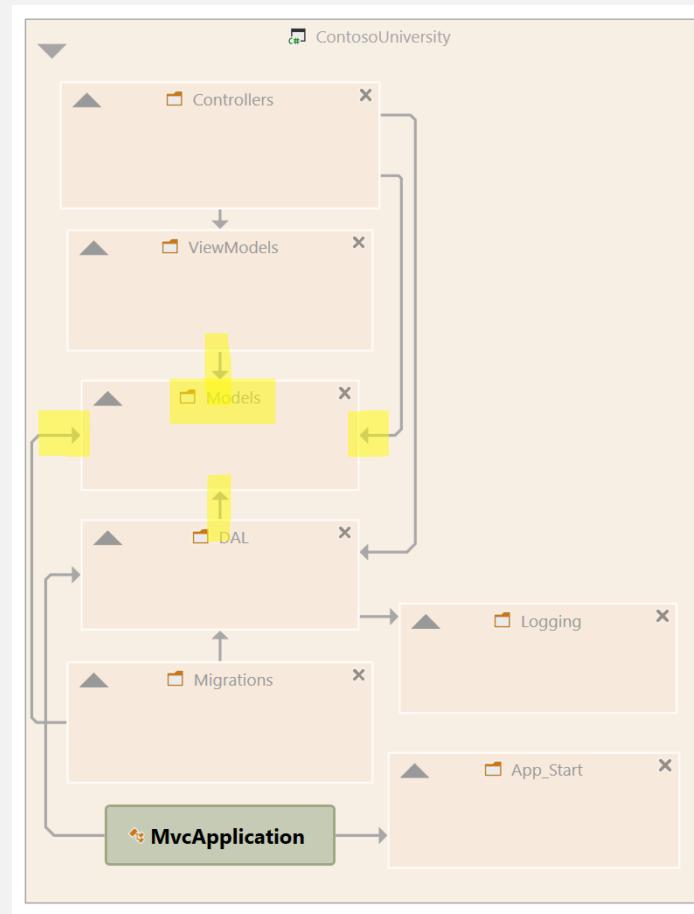
Start at the center and work outwards

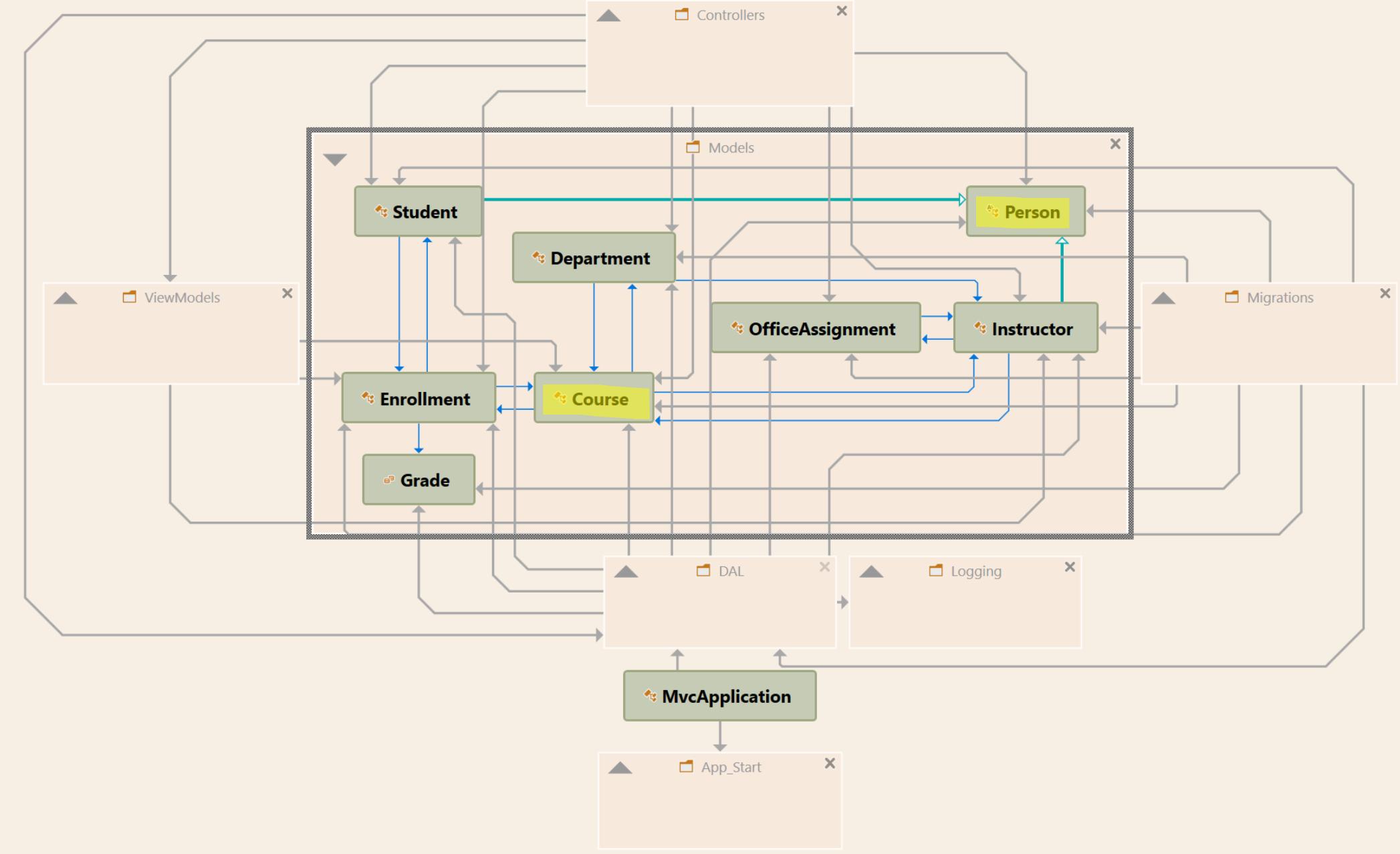
ReSharper Type Hierarchy Diagram

Visual Studio architecture tools

NDepend

...





Per class...

- Enable NRT (`#nullable enable`)
- For every property and constructor parameter
 - Find all *write* usages
 - If potentially set to null, annotating with `?` is needed



Adding #nullable enable

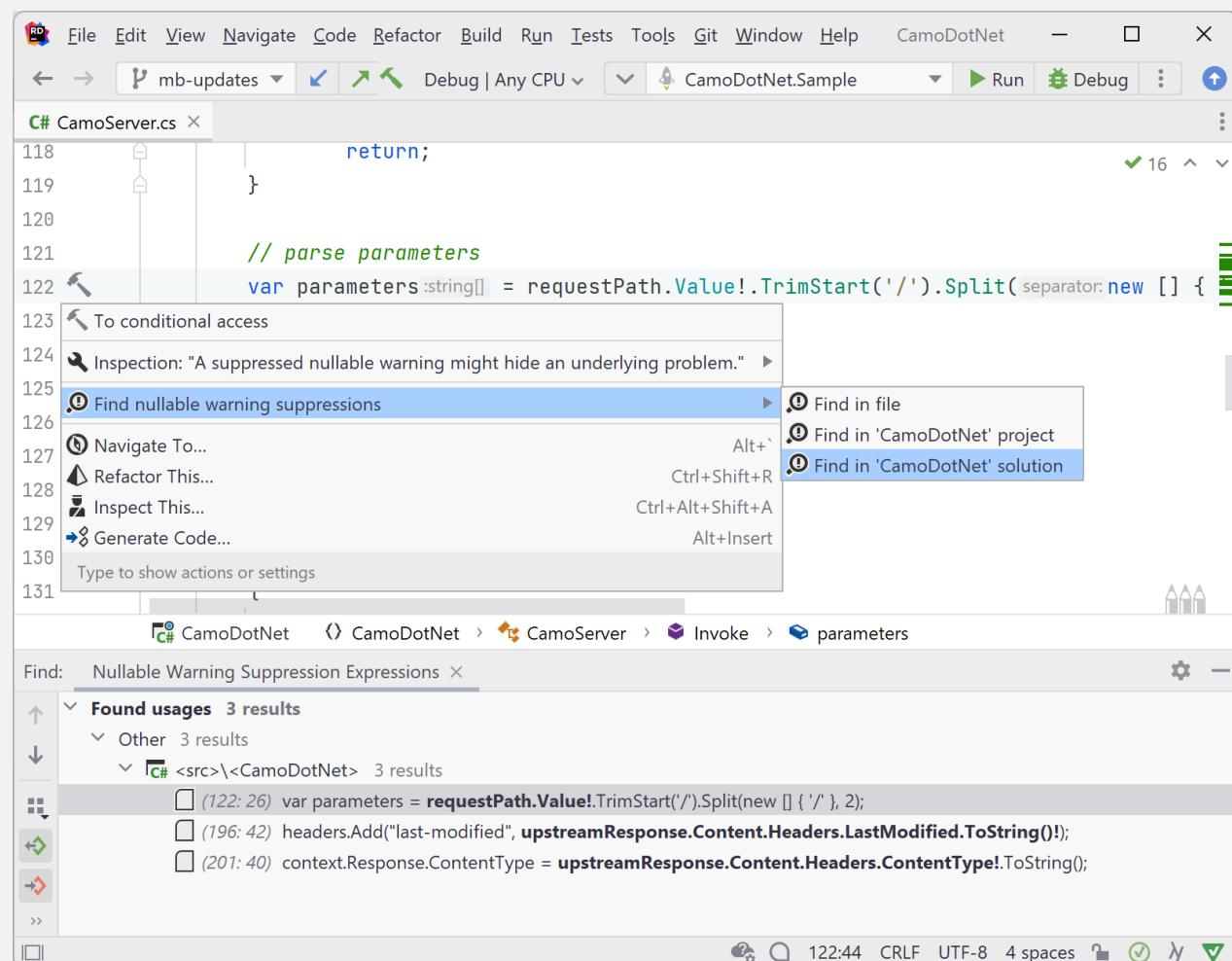
Demo

Annotate, and redesign!

- Sometimes redesign makes more sense
- Nullable annotations surface through code base (like `async/await`)
 - Do you want to check for null at every current and future usage?
 - Or do it once and return a “[null object](#)”?
- “It depends”

Suppressions should be temporary

- Temporarily disable flow analysis
 - “what if this was non-null”?
- Remove suppressions



Don't be afraid of null

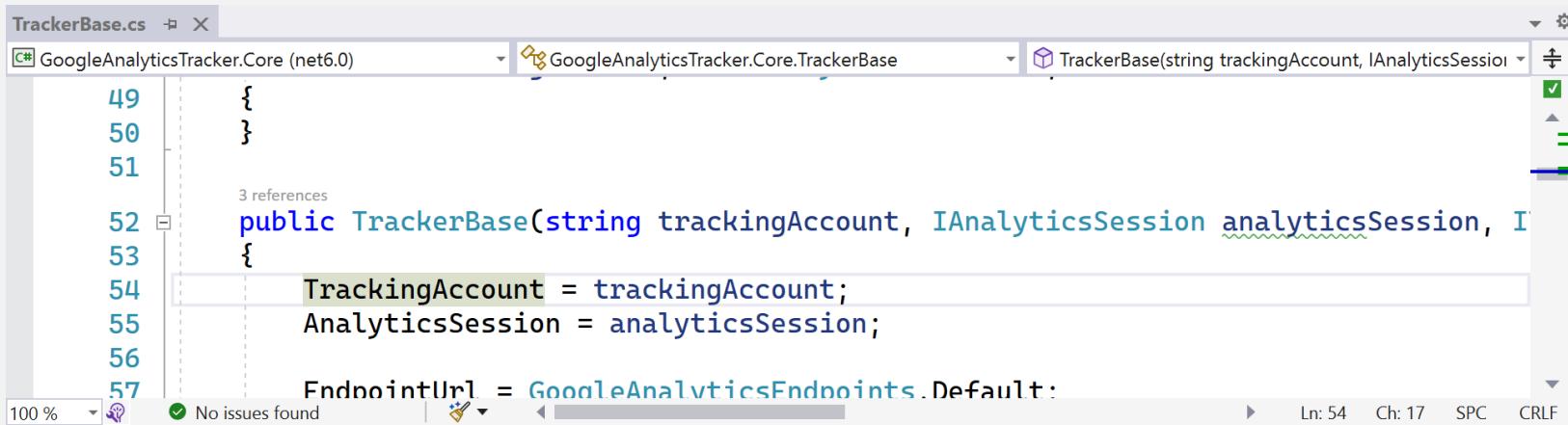
-  Gain more confidence in compiler/IDE flow analysis
-  Completely get rid of all null usages in code
- Build a safety net!
 - Annotate, suppress, redesign – passing null is totally fine
 - Know where it is potentially passed
 - Reduce chance of NullReferenceException

Summary: Techniques

- Always enable NRT for new projects
- Start at the center and work outwards
- Annotate, and redesign
- Suppressions should be temporary
- Don't be afraid of null

Tools that can help

Value tracking (origin/destination)



The screenshot shows the Visual Studio IDE with the code editor open to `TrackerBase.cs`. The code defines a constructor for `TrackerBase` that takes `trackingAccount`, `analyticsSession`, and `endpoints` parameters. The `trackingAccount` parameter is highlighted in yellow, indicating it is being tracked. The code editor interface includes tabs for the project and file, a status bar at the bottom, and a navigation bar above the code.

```
49     {
50 }
51
52     public TrackerBase(string trackingAccount, IAnalyticsSession analyticsSession, I
53     {
54         TrackingAccount = trackingAccount;
55         AnalyticsSession = analyticsSession;
56
57         EndpointUrl = GoogleAnalyticsEndpoints.Default:
```



```
public void ConfigureServices(IServiceCollection services)
{
    services.AddRouting(options => options.Lov
    services.Configure<CookiePolicyOptions>(options =>
    {
        // This lambda determines whether user consent is required for a given request.
        options.CheckConsentNeeded = _ => true;
        options.MinimumSameSitePolicy = SameSiteMode
    });

    services.AddGoogleAnalyticsTracker(options =>
    {
        // ReSharper disable once StringLiteralTypo
        options.TrackerId = "UA-XXXXXX-XX";
        options.ShouldTrackRequestInMiddleware = true;
    });
}
```



JetBrains Annotations to C# annotations

The screenshot shows an IDE interface with the following code snippet:

```
#nullable enable

[NotNull]
    9 usages  Maarten Balliauw
private static string ReadColumnFromCsv(
    CsvReader csv,
    ...,
    [CanBeNull] string columnName,
        defaultValue = "")

    NullOrEmpty(columnName)
ame]
e;

    Inspection: "Use type annotation syntax" ▶
        Use type annotation syntax
        Create overload without parameter
        Not null
        Unknown nullability
        Inspection: "Use type annotation syntax" ▶
            Navigate To...          Alt+` 
            Refactor This...       Ctrl+Shift+R
            Inspect This...        Ctrl+Alt+Shift+A
            Generate Code...       Alt+Insert
Type to show actions or settings
```

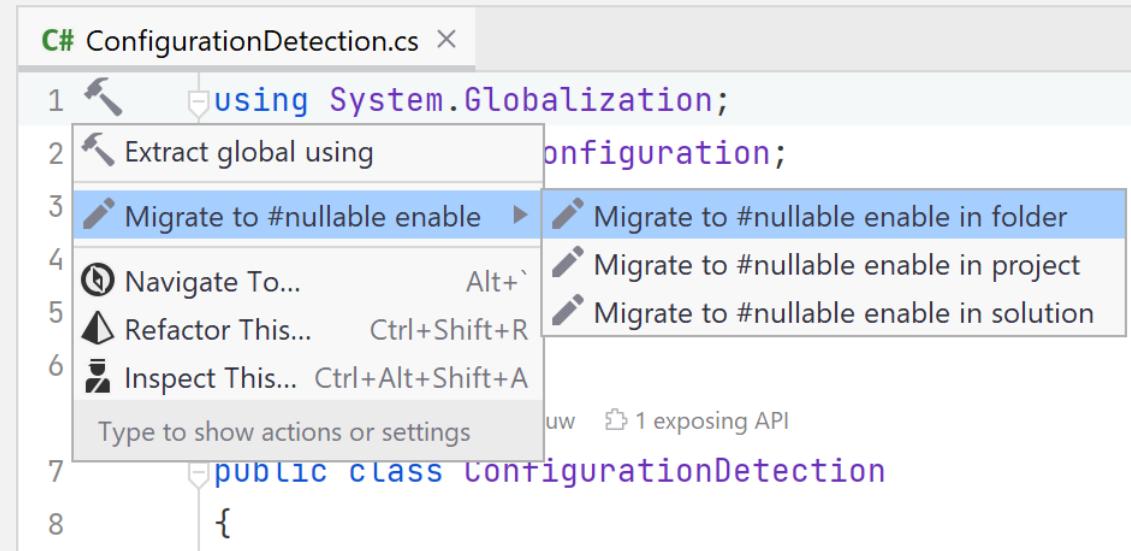
A context menu is open at the bottom of the code editor, specifically over the `[CanBeNull]` annotation. The menu items are:

- Use type annotation syntax
- Create overload without parameter
- Not null
- Unknown nullability
- Inspection: "Use type annotation syntax" ▶
- Navigate To... Alt+`
- Refactor This... Ctrl+Shift+R
- Inspect This... Ctrl+Alt+Shift+A
- Generate Code... Alt+Insert

At the bottom of the menu, there is a placeholder text: "Type to show actions or settings".

Automatic migration*

- Inserts [NotNull] and [CanBeNull] from base classes
- Infers annotations by looking at usages
 - If null anywhere, gets annotated



*Not a silver bullet, but a great help

Infer from existing null checks

- Your code has clear hints about nullability
- Quiz: should we annotate?

```
public static string ReadColumn(  
    string columnName,  
    string defaultValue = "")  
{  
    if (columnName != null)  
    {  
        if (_mappings.TryGetValue(columnName, out var columnIndex)  
            && _data.TryGetValue(columnIndex, out var columnData))  
        {  
            return columnData ?? defaultValue;  
        }  
    }  
  
    return defaultValue;  
}
```



Third-party libraries

- .NET BCL and some libraries are annotated – great!
- Some libraries use JetBrains.Annotations – great!
- Many ^(most 😞) are *not* annotated
 - Have to assume null everywhere
 - R#/Rider – try pessimistic analysis
https://www.jetbrains.com/help/resharper/Code_Analysis__Value_Analysis.html#modes

(De)serializing JSON

How do you go about this warning?

```
[JsonProperty("name")]
```

```
public string Name { get; set; }
```

Non-nullable property 'Name' is uninitialized. Consider declaring the property as nullable.

```
[JsonProperty("name")]
```

```
public string Name { get; set; }
```

```
in class User
```

⋮

(De)serializing JSON

- // 1
[JsonProperty("name")]
public string? Name { get; set; }
 - // 2
[JsonProperty("name")]
public string Name { get; set; } = default!;
 - // 3
[JsonProperty("name")]
public string Name { get; set; } = "Unknown";
 - // 4
private readonly string _name;

[AllowNull]
[JsonProperty("name")]
public string Name
{
 get => _name;
 init => _name = value ?? "Unknown";
}
- Loses nullability information
- antipattern (incorrect information)**
- Good alternative
- Good (cumbersome) alternative

(De)serializing JSON

- ```
// 5
[JsonProperty("name")]
public required string Name { get; set; }
```

Personal recommendation\*

\*Your JSON data may still be null...  
Checking input in the property or using  
IJsonOnDeserialized may be helpful.

# Be careful with Entity Framework Core

*A **property** is considered optional if it is valid for it to contain null. If null is not a valid value to be assigned to a property then it is considered to be a required property. When mapping to a relational database schema, required properties are created as non-nullable columns, and optional properties are created as nullable columns.*

**TL;DR:** When you enable NRT in a project or file, your database schema may change unexpectedly.

<https://docs.microsoft.com/en-us/ef/core/modeling/entity-properties?tabs=data-annotations%2Cwithout-nrt#:~:text=A%20property%20is,as%20nullable%20columns>

<https://docs.microsoft.com/en-us/ef/core/miscellaneous/nullable-reference-types#required-and-optional-properties>

# Summary: Tools

- Value tracking (value origin/destination)
- Convert JetBrains Annotations to C# nullable annotations
- Automatic migration
- Infer from existing null checks
- Third-party libraries...
  - JSON
  - Entity Framework Core

# Summary



# Nullable reference types in C#

- Design-time and compile-time safety net. Not runtime!
- Nullable annotation context to determine level of help
- Annotations to communicate intent  
(`null` is fine if you know where)
- Null-forgiving operator is an **antipattern**
- Tools and techniques to migrate

A wide-angle photograph of a natural landscape at sunset. The sky is filled with dramatic, colorful clouds ranging from deep orange to dark grey. Below the sky, a calm river or lake curves through a dense forest of green trees. In the foreground, there's a grassy area. On the right side of the image, a man with short hair and glasses, wearing a dark polo shirt, is smiling and looking towards the camera.

# Thank you!

<https://blog.maartenballiauw.be>

<https://t.ly/WMLC>

<https://mastodon.online/@maartenballiauw>

