Safer Code

Nullability and Null Operators in C#

Jeremy Clark
www.jeremybytes.com
@jeremybytes

Nullability in C#

- What Nullability Is
- What Nullability Is Not
- Nullability Operators
  - `?.` and `?[]`
  - `!`
  - `??` and `??=`

Value Types vs. Reference Types

Value Types

- Stored on the stack
- Cannot be null*
- Default is "bitwise zero"
  - int = 0; bool = false
- Ex: int, bool, enum, struct

Reference Types

- Stored on the heap (with memory address in the stack)
- Cannot be forced non-null*
- Default is "null"
- Ex: string, List<int>, class

Enabling Nullability

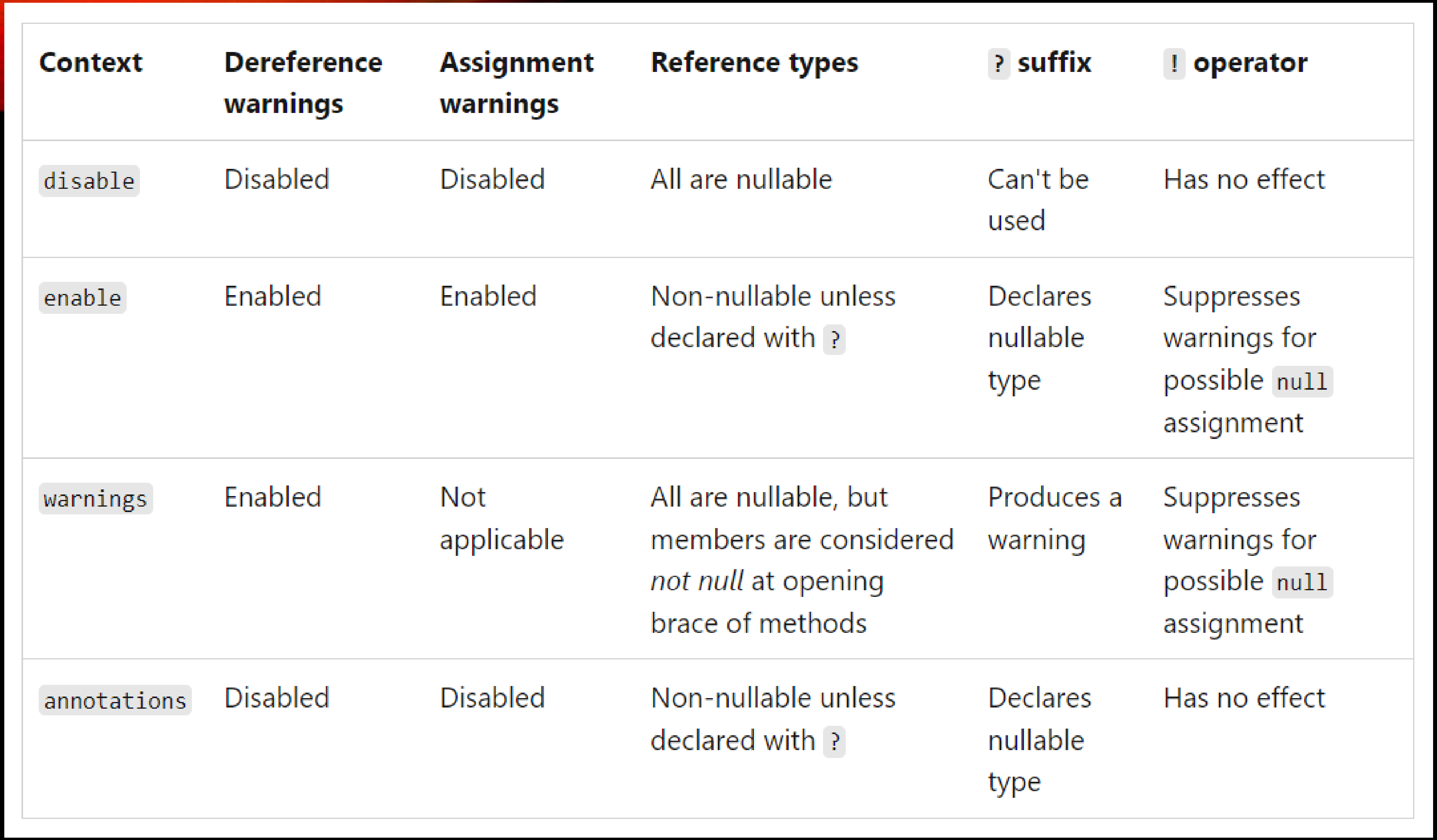
- Project Level

```
<Nullable>enable</Nullable>
```

- Code Level

```
#nullable enable
```

Nullable Contexts



Context	Dereference warnings	Assignment warnings	Reference types	`?` suffix	`!` operator
`disable`	Disabled	Disabled	All are nullable	Can't be used	Has no effect
`enable`	Enabled	Enabled	Non-nullable unless declared with `?`	Declares nullable type	Suppresses warnings for possible `null` assignment
`warnings`	Enabled	Not applicable	All are nullable, but members are considered *not null* at opening brace of methods	Produces a warning	Suppresses warnings for possible `null` assignment
`annotations`	Disabled	Disabled	Non-nullable unless declared with `?`	Declares nullable type	Has no effect

https://learn.microsoft.com/en-us/dotnet/csharp/nullable-references#nullable-contexts

Marking Types as Nullable

- With nullability enabled, reference types are non-nullable by default.
- Nullable types must be marked with '?'.

```
    Person firstPerson; // non-null
    Person? secondPerson; // nullable
```

Starting with .NET 6, new projects have nullability enabled by default.

What Nullability Is

- A way to get *compile-time* warnings about possible null references.

- A way to make the intent of your code more clear.

What Nullability Is Not

- NOT a way to prevent null reference exceptions at runtime.
- NOT a way to prevent someone from passing a null to your method.
- NOT a way to prevent someone from assigning a null to an object.

Null Conditional Operators

- `? and ?[]`
- Ex: `tokenSource?.Cancel();`
  - If “tokenSource” is not null, “Cancel()” is called.
  - If “tokenSource” is null, “Cancel()” is *not* called.

Note: the null check is thread-safe.

Null Forgiving Operator

- !
- If the compiler issues an incorrect warning, the "!" can be used to suppress the warning.
- Ex: `task.Exception!.Flatten()`

Null Coalescing Operator

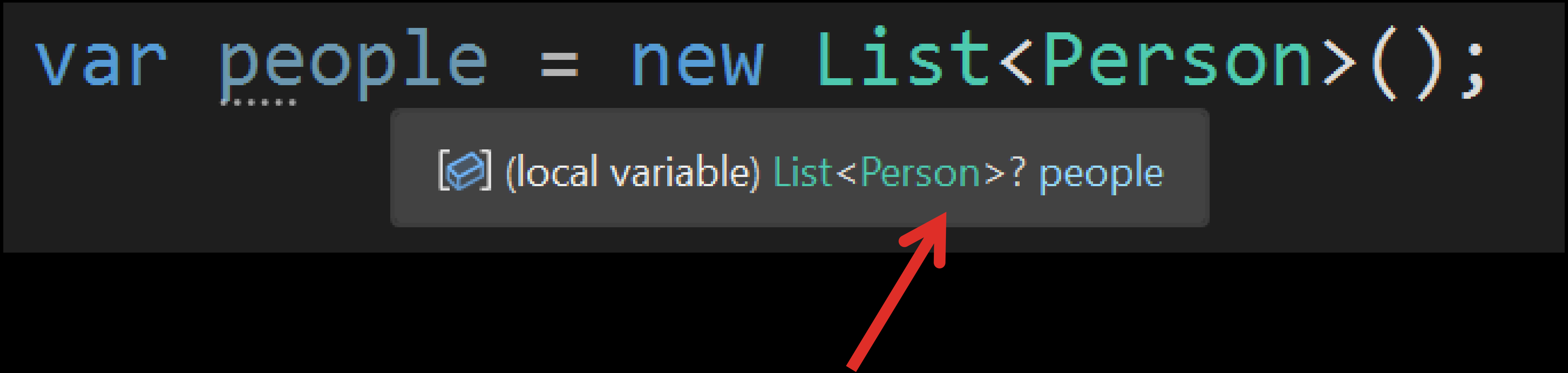
- `??`
- Can be used to provide an alternate value if something is null.
- Ex: `return people ?? new List<Person>();`
  - If "people" is not null, it is returned.
  - If "people" is null, a new empty list is returned.

Null Coalescing Operator

- `??=`
- Can be combined with “=“ to do coalescing and assignment at the same time.
- Ex: `people ??= new List<Person>();`
  - If “people” is not null, the value is unchanged.
  - If “people” is null, an empty list is assigned.

Important Note about “var”

- Using “var” results in a nullable type.



```
var people = new List<Person>();
```

(local variable) List<Person>? people

©Jeremy Clark 2023

What Nullability Is

- A way to get *compile-time* warnings about possible null references.

- A way to make the intent of your code more clear.

What Nullability Is Not

- NOT a way to prevent null reference exceptions at runtime.
- NOT a way to prevent someone from passing a null to your method.
- NOT a way to prevent someone from assigning a null to an object.

Nullability Operators

`?. / ?[]` (Null Conditional Operators)

```
tokenSource?.Cancel();
```

`!` (Null Forgiving Operator)

```
task.Exception!.Flatten().InnerExceptions
```

`?? / ??=` (Null Coalescing Operators)

```
return result ?? new List<Person>();
```

Resources

Code Samples & Resources

https://github.com/jeremybytes/sdd-2023

Thank You!

Jeremy Clark

- www.jeremybytes.com
- jeremy@jeremybytes.com
- @jeremybytes

https://github.com/jeremybytes/sdd-2023