NULLABILITY IN C# 8

JARED PARSONS

INTRODUCTION

- Working in programming languages at Microsoft for ~13 years
- Developer lead on the C# compiler
- Member of the C# language design team

DESIGNING NULLABILITY

Thinking about nullability in the landscape of C#

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

C# AND NULL REFERENCES

- C# 7.3 has no null reference tracking
  - Developers instead follow best practices
  - Read API documentation, guard APIs they're unsure of
  - Allocate time to fix all the crash reports they get
- C# 8.0 wants to fix this
  - But how do you add nullable tracking to a 20 year old language?

CREATING VS. EXTENDING

- Creating a language has unbounded options
  - Nothing to be compatible with
  - Change the type system, API focus, etc …
- Extending a language narrows the possible solutions
  - Existing features that never thought about nullability
  - Existing libraries that want safety but want to remain compatible
  - Must maintain the look and feel of the language

WHAT ABOUT A PERFECT TYPE SYSTEM?

- Perfectly separates nullable and non-nullable types
- C# variants exist with "perfect" null safety
  - Spec#, Sing# and several internal experiments
  - Successful at implementing null safety
  - Unsuccessful at adoption because it simply wasn't C# anymore
- Breaks lots of existing patterns
  - Array creation like new string[4]
  - Calling methods in constructors
  - Using default(T) in generics

WHAT ABOUT AN API SOLUTION?

- Solution: Optional<T>
  - Nullable<T> but for reference types
  - Value cannot be used directly
  - Must do an explicit null check to get value
- Problem
  - Does nothing for existing code
  - Breaks binary compatibility
  - It's a yet another form of null

COMPONENTS FOR SUCCESS

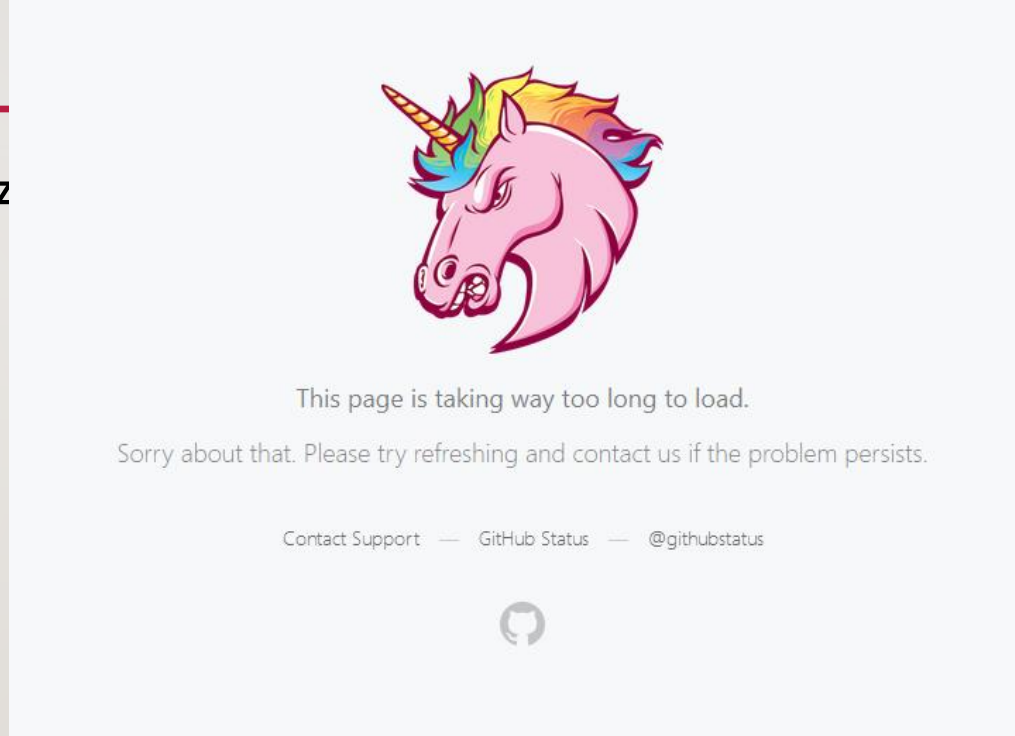
- Must maintain the look and feel of C#
- Add value to existing code without major rewrite
- Embrace the imperfect history of C#

LOTS OF C# CODE TO ANNOTATE

- Code bases of varying size
  - Small: < 10,000 lines
  - Medium: < 100,000 lines
  - Large: < 1,000,000 lines
  - Jumbo: > 1,000,000 lines
- Roslyn GitHub repository as an example has ~5,000,000 lines
- Null safety must be usable in all these code bases

CAN'T ENABLE IN ONE CHANGE

- Cost of adopting nullability proportional to code base siz
  - Medium code bases can change hundreds of files
  - Large code bases can change thousands
- Pull Requests of this size are problematic
  - GitHub won't display them well (if at all)
  - Reviewing is time consuming
- Incremental adoption
  - Annotate a single component, file or directory
  - Keep PRs small, singe focused and reasonably sized



.NET HAS A BROAD ECOSYSTEM

- Ecosystem
  - NuGet.org has ~175,000 unique packages
  - MyGet.org has thousands of customers
  - Private companies with proprietary libraries
- Developers want better null checking
  - Can't wait for all dependencies to move first
  - Need to adopt independent of dependencies

NON-NULLABLE REFERENCE TYPES

AKA the last two years of the C# team

PRINCIPLES

1. Non-Nullable Reference types are the default
2. Maintain the look and feel of C#
3. Leverage existing null checking patterns
4. Don’t impact runtime or semantic behavior

NULL BEHAVIORS

- Issue Warning
  - Assigning or passing null to non-nullable reference type
  - Assigning or passing nullable reference type to non-nullable one
  - Using default(T) for a non-nullable reference type
  - Using an instance method or field on a possibly null reference type
  - Failing to assign non-nullable field in class constructor
- No Warning
  - Creating an array of a non-nullable reference type: new string[14]
  - Creating a struct where field is non-nullable reference type

REVIEW

- The ? annotation allows developers to express intent around null
- The compiler recognizes and learns from existing null checks
- Framework annotations push your code to null safety

STATE OF NULLABILITY IN .NET

Unhandled exception. System.NullReferenceException: Object reference not set to an instance of an object.

.NET CORE 3.0

- Ships C# 8 with support for nullable reference types
- .NET SDK partially null annotated
  - Corelib has nullable annotations (10K additions)
  - Remaining SDK largely unannotated
  - Temporary state due to time limitations
  - https://github.com/dotnet/corefx/issues/40623
- .NET Ecosystem partially annotated
- .NET Desktop is not annotated

.NET AND NULLABILITY GOING FORWARD

- Nullable Rollout Phase
  - Time between .NET Core 3.0 and .NET 5
  - Goal is getting all relevant parts of .NET SDK Annotated
- Encouraging ecosystem to add annotations in this timeframe
- Affect on early adopters of nullability
  - Will likely see new warnings as annotations are added
  - Will be proportional to the APIs consumed

RECOMMENDATION FOR LIBRARY AUTHORS

- Annotate libraries during the nullable rollout phase
- Consider multi-targeting
  - .NET Core 3.0 for nullable annotations
  - .NET Standard 2.0 for reach
  - Enable nullable warnings for .NET Core 3.0
  - Disable nullable warnings for .NET Standard 2.0

ADOPTION STRATEGIES

- Annotate entire code base at once
  - Recommend limiting to 100 files
- Dependency Order
  - Start root project and move outwards
  - Annotate one project until done then move to next
- Annotate as changed
  - Every bug fix must add #nullable to files it changes

ADOPTION EXAMPLES

- All at once
  - vs-threading 89 files
- Divide and Conquer
  - .NET SDK 3.0 annotations
  - .NET SDK 5.0 annotations
- Annotate as changed
  - C# compiler + IDE annotations

MORE RESOURCES (1)

Blog posts:

- Try out Nullable Reference Types: devblogs.microsoft.com/dotnet/try-out-nullable-reference-types
- Take C# 8.0 for a spin: devblogs.microsoft.com/dotnet/take-c-8-0-for-a-spin
- Nullable Reference Types in C#: devblogs.microsoft.com/dotnet/nullable-reference-types-in-csharp

MORE RESOURCES (2)

- NRTs overview: docs.microsoft.com/dotnet/csharp/nullable-references
- Nullable attributes: https://docs.microsoft.com/en-us/dotnet/csharp/nullable-attributes
- Update libraries to NRTs: docs.microsoft.com/dotnet/csharp/nullable-attributes
- NRTs tutorial: docs.microsoft.com/dotnet/csharp/tutorials/nullable-reference-types
- Migrate an app to NRTs: docs.microsoft.com/dotnet/csharp/tutorials/upgrade-to-nullable-references

Q & A

- jaredpar@microsoft.com
- https://github.com/jaredpar
- https://twitter.com/jaredpar