C# 7

Kathleen Dollard

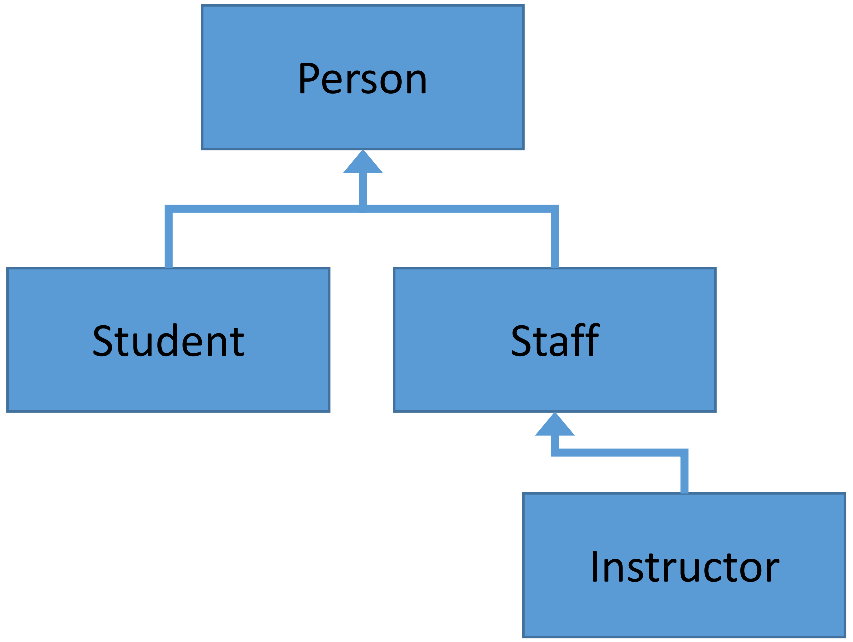
Twitter: @KathleenDollard

kathleen.a.dollard@gmail.com

Demo!

C# 7 - The Little Things

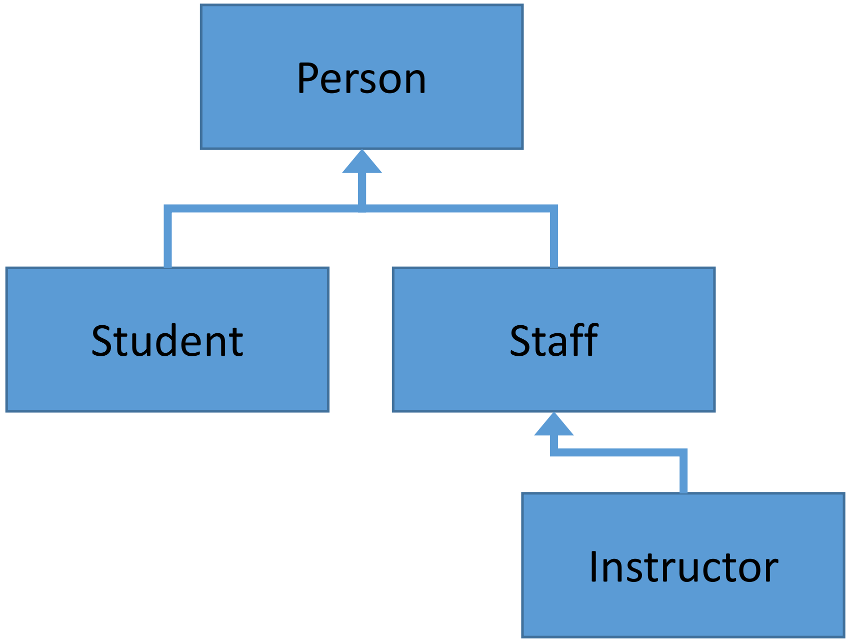
Demo Hierarchy



Object oriented probably better in this case – area is an intrinsic aspect

```
switch(shape)
{
    case Circle c:
        WriteLine($"circle with radius {c.Radius}");
        break;
    case Rectangle s when (s.Length == s.Height):
        WriteLine($"{s.Length} x {s.Height} square");
        break;
    case Rectangle r:
        WriteLine($"{r.Length} x {r.Height} rectangle");
        break;
    default:
        WriteLine("<unknown shape>");
        break;
    case null:
        throw new ArgumentNullException(nameof(shape));
}
```

In this demo, messages aren't intrinsic



Winter 2016

- Thanks… congrats to hockey team

Spring 2016

- Thanks… sorry about the flood

Summer 2016

- Thanks.. we will find the pranksters

Fall 2016

- Thanks… what beautiful weather we had

Multiple Return Tricks to Stop Doing

- Out parameters:
  - Use is clunky (even with the improvements described above), and they don’t work with async methods.
- System.Tuple<...> return types:
  - Verbose to use and require an allocation of a tuple object.
- Custom-built transport type for every method:
  - A lot of code overhead for a type whose purpose is just to temporarily group a few values.
- Anonymous types returned through a dynamic return type:
  - High performance overhead and no static type checking.

Cool Stuff I Didn’t Show

- Ref return types
  - Useful for massive array and structure manipulation
- Generalized async return types
  - ValueTask<T>
  - Mostly used inside the tools
- More expression body members
  - Constructors, finalizers, accessors (get/set)

What about future versions of C#?

github.com/dotnet/roslyn/blob/master/docs/Language%20Feature%20Status.md

C# 7.1 (proposed)

- Reference assemblies
  - Mostly for IDE and tooling purposes
- Default expressions
  - Default for default ;-) allow type of default to be inferred
- MainAsync
  - Makes playing with async in console apps cleaner
- Tuple projection initializers (infer tuple names
  - (x.f1, x?.f2) same as (f1: x.f1, f2: x?.f2)
- Loosening pattern matching rules for generics
  - https://github.com/dotnet/roslyn/pull/18784
- private protected

C# 7.1 (proposed)
Reference Assemblies

- Lightweight versions of metadata-only assemblies

C# 7.1 (proposed) DefaultExpressions

- Make it easier to use default

```
int x = default(int);                 int x = default;
...                                   ...
y = default(Func<Task<List<T>>>);     y = default;
```

C# 7.1 (proposed)
Main Async

- Allows console apps to be async
- This is almost exclusively to make it less confusing to play with and learn async/await

C# 7.1 (proposed)
Infer Tuple Names

```
cust = new Customer(“Joe”, “Jones”);
t = (cust.FirstName, cust.LastName);
Console.WriteLine(t.FirstName, t.LastName);
```

- Follows the same rules as anonymous type inferred names, with a few exceptions
- Spec includes VB rules

(proposed)

Pattern Matching Fix

```
public void DoSomething<T>(T p)
// where T : A
{
    var x = p as A;
    if (x != null) { }

    if (p is A x2) { }
    var y = (A)p;
}

public class A { }
// mostly likely to see when using derived
public class DerivedFromA : A { }
```

Gives error because no cast

We understand this failing

Visual Studio 2-17 Update 3 Preview

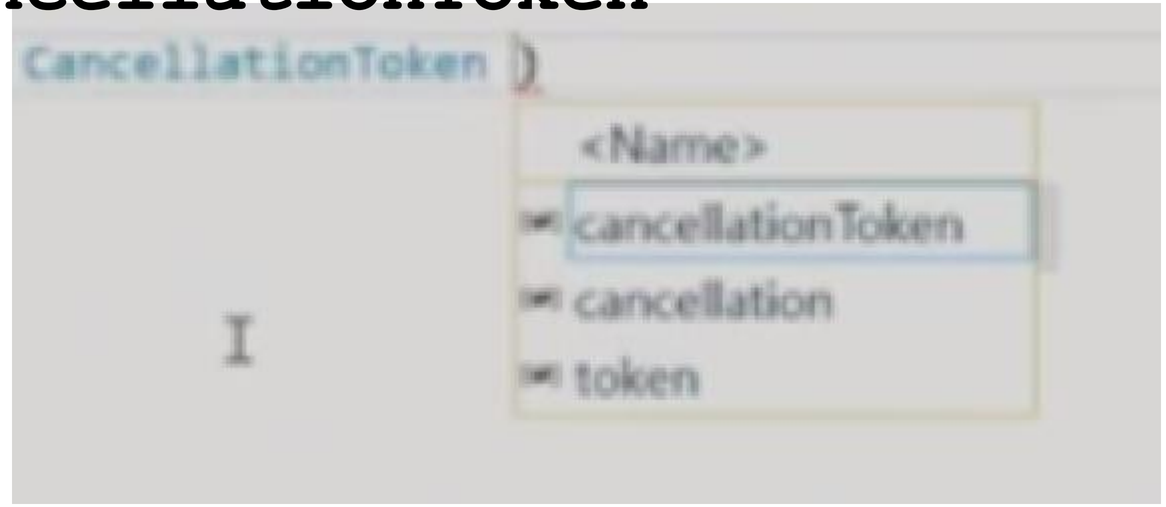
- Name suggestions

```
...(CancellationToken
```

Visual Studio 2-17 Update 3 Preview

- Name suggestions

```
...(CancellationToken
```



```
CancellationToken )
```

C# 7.2 (proposed)
Make low level/unsafe scenarios better

- ref readonly (in parameters)
  - For value types: perf/size of pass by ref, immutability of pass by val
  - Also return readonly ref and possible use in readonly structs
- blittable
  - Value types with no contained ref types
  - Makes interop and some other scenarios easier
- interior pointer
  - Safety for Span<T>

Span<T> is a small, but critical, building block for a much larger effort to provide .NET APIs to enable development of high scalability server applications.

C# 8.0 (proposed)

- Default interface methods

- Nullable reference types – static null checking

C# 8.0 (proposed)
Default interface methods

- The syntax for an interface is extended to permit
  - a body for a method or indexer, property, or event accessor (i.e. a "default" implementation)
  - static methods, properties, indexers, and events.
  - Explicit access modifiers (the default access is public)
  - override modifiers
- Implementation for classes and structs without an overriding implementation
- Interfaces may not contain instance state
  - Instance fields and auto properties not allowed since they supply instance state
- Static fields are permitted,
- Static and private methods allowed for organization

C# 8.0 (proposed)
Default interface methods

```
public interface IAnInterface
{
    void LotsOfStuff();
    string Name
    {
        get { return ""; }
    }
}
```

C# 8.0 (proposed)
Default interface methods

```
public interface IAnInterface
{
    void LotsOfStuff();
    string Name
    {
        get { return ""; }
    }
}
```

```
public interface IAnInterface
{
    void LotsOfStuff();
}
```

Previous

C# 8.0 (proposed)
Default interface methods

- Use/Abuse
  - Traits and Mixins
  - Which point to multiple inheritance

C# 8.0 (proposed)
Nullable Reference Types

- Huh?
  - Don’t they know reference types are already nullable?
- Hard to add true non-nullable types
  - Add mechanism by which you can declare nullable vs non-nullable types
  - Static analysis can then find many cases of misuse
- So,
  - null reference errors become numerous different errors
  - due to lack of initialization

C# 8.0 (proposed)

- Default interface methods
  - Interop with Android (Java) and Ios (Swift)
  - Interface has default implementation, virtual extension methods
  - Scope, static, inheritance, mostly supported
  - No instance state allowed
  - Probably runtime dependent
- Nullable reference types – static null checking
  - Null ref errors remain the most common
  - T and T? differ only in the warnings the compiler provides

Backwards compatibility

- What I wrote before works in the new version

Forwards compatibility

- What I write now works in previous versions

Questions?

Kathleen Dollard

Twitter: @KathleenDollard

kathleen.a.dollard@gmail.com