# The Road to C# 8

**Kathleen Dollard**

.NET Team, Microsoft

@kathleendollard

Kathleen.Dollard@Microsoft.com

# Themes

**Performance, special scenarios**

- Span<T>
- Ref goodness
- Async streams

**Productivity**

- Very long list

*These features make life better*
*even if you don’t use them directly*

# Demo

# Type Specific Behavior – OO vs Patterns

- Change code in class when…
  - You have access to the code
  - The change is intrinsic to the type
  - Example: The area of a geometric shape
- Use extension methods when…
  - You can’t access the code
  - The change is intrinsic to the type
- Use pattern matching when…
  - The change is not intrinsic to the type
  - Example: Each semester a school sends messages to its students and staff

# Demo

# Nullable Reference Types

Yes, we are changing the default nullability for types in C#

Only if you opt in
Only as warnings

This is entirely a compiler feature,
no frameworks were harmed in the creation of this feature

Because, NullReferenceException is most common exception

# Demo

# Some Older Things I Didn’t Cover in Demo

- private protected
- Tuple name inference
- Improved overload resolution
- Numeric separators (underscores)
- Binary literals
- ref goodness
- Span<T>

# Later Previews

# Recursive patterns

```
return (p.MiddleName != null)
    ? $"{p.FirstName} {p.MiddleName[0]}. {p.LastName}"
    : $"{p.FirstName} {p.LastName}";
```

# Recursive patterns

```
switch (p.FirstName, p.MiddleName, p.LastName)
{
    case (string f, string m, string l):
        return $"{f} {m[0]}. {l}";
}
```

# Recursive patterns

```
switch (p.FirstName, p.MiddleName, p.LastName)
{
    case (string f, string m, string l):
        return $"{f} {m[0]}. {l}";
    case (string f, null, string l):
        return $"{f} {l}";
}
```

# Recursive patterns

```
switch (p.FirstName, p.MiddleName, p.LastName)
{
    case (string f, string m, string l):
        return $"{f} {m[0]}. {l}";
    case (string f, null, string l):
        return $"{f} {l}";
    case (string f, string m, null):
        return $"{f} {m}";
}
```

# Recursive patterns

```
switch (p.FirstName, p.MiddleName, p.LastName)
{
    case (string f, string m, string l):
        return $"{f} {m[0]}. {l}";
    case (string f, null, string l):
        return $"{f} {l}";
    case (string f, string m, null):
        return $"{f} {m}";
    case (string f, null, null):
        return f;
}
```

# Recursive patterns

```
switch (p.FirstName, p.MiddleName, p.LastName)
{
    case (string f, string m, string l):
        return $"{f} {m[0]}. {l}";
    case (string f, null, string l):
        return $"{f} {l}";
    case (string f, string m, null):
        return $"{f} {m}";
    case (string f, null, null):
        return f;
    case (null, string m, string l):
        return $"Ms/Mr {m[0]}. {l}";
}
```

# Recursive patterns

```
switch (p.FirstName, p.MiddleName, p.LastName)
{
    case (string f, string m, string l):
        return $"{f} {m[0]}. {l}";
    case (string f, null, string l):
        return $"{f} {l}";
    case (string f, string m, null):
        return $"{f} {m}";
    case (string f, null, null):
        return f;
    case (null, string m, string l):
        return $"Ms/Mr {m[0]}. {l}";
    case (null, null, string l):
        return $"Ms/Mr {l}";
    case (null, string m, null):
        return $"Ms/Mr {m}";
    case (null, null, null):
        return "Someone";
}
```

# Switch expressions

```
return (p.FirstName, p.MiddleName, p.LastName) switch
{
    (string f, string m, string l) => $"{f} {m[0]}. {l}",
    (string f, null    , string l) => $"{f} {l}",
    (string f, string m, null    ) => $"{f} {m}",
    (string f, null    , null    ) => f,
    (null    , string m, string l) => $"Ms/Mr {m[0]}. {l}",
    (null    , null    , string l) => $"Ms/Mr {l}",
    (null    , string m, null    ) => $"Ms/Mr {m}",
    (null    , null    , null    ) => "Someone"
};
```

# Implicitly typed new-expressions

```
Person[] people =
        {
            new         ("Carl", "Scott", "Hunter"),
            new         ("Neal", "M", "Gafter"),
            new         ("Dustin", "Campbell"),
            new         ("Miguel", "de Icaza")
        };
```

# Implicitly typed new-expressions

```
Person[] people =
        {
            new ("Carl", "Scott", "Hunter"),
            new ("Neal", "M", "Gafter"),
            new ("Dustin", "Campbell"),
            new ("Miguel", "de Icaza")
        };
```

# Default Interface Members

```
interface ILogger
{
    void Log(LogLevel level, string message);
}


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

# Default Interface Members

```
interface ILogger
{
    void Log(LogLevel level, string message);
    void Log(Exception ex)
}

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

# Default Interface Members

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

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

# Default Interface Members

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

class TelemetryLogger : ILogger
{
    // send message
    public void Log(LogLevel level, string message) { }

    // capture crash dump and send message
    public void Log(Exception ex) { }
}
```

# Places to go, things to do…

- Kathleen Dollard, Microsoft
  - @kathleendollard
  - Kathleen.Dollard@Microsoft.com
- Learn more about C# 8.0
  - https://aka.ms/csharp8
- Get the previews, and send us feedback!
  - https://blogs.msdn.microsoft.com/dotnet announcement
  - github.com/dotnet/csharplang/wiki
- Check the docs:
  - docs.microsoft.com/en-us/dotnet/csharp/whats-new/