

# C# Extension Methods

---

## Getting Started with Extension Methods



**Elton Stoneman**

Consultant & Trainer

@EltonStoneman   [blog.sixeyed.com](http://blog.sixeyed.com)

```
var strings = new List<string> { "a", "b", "c" };  
Assert.AreEqual(3, strings.Count());
```

Using extension methods

**Count()** is an extension

**System.Linq**

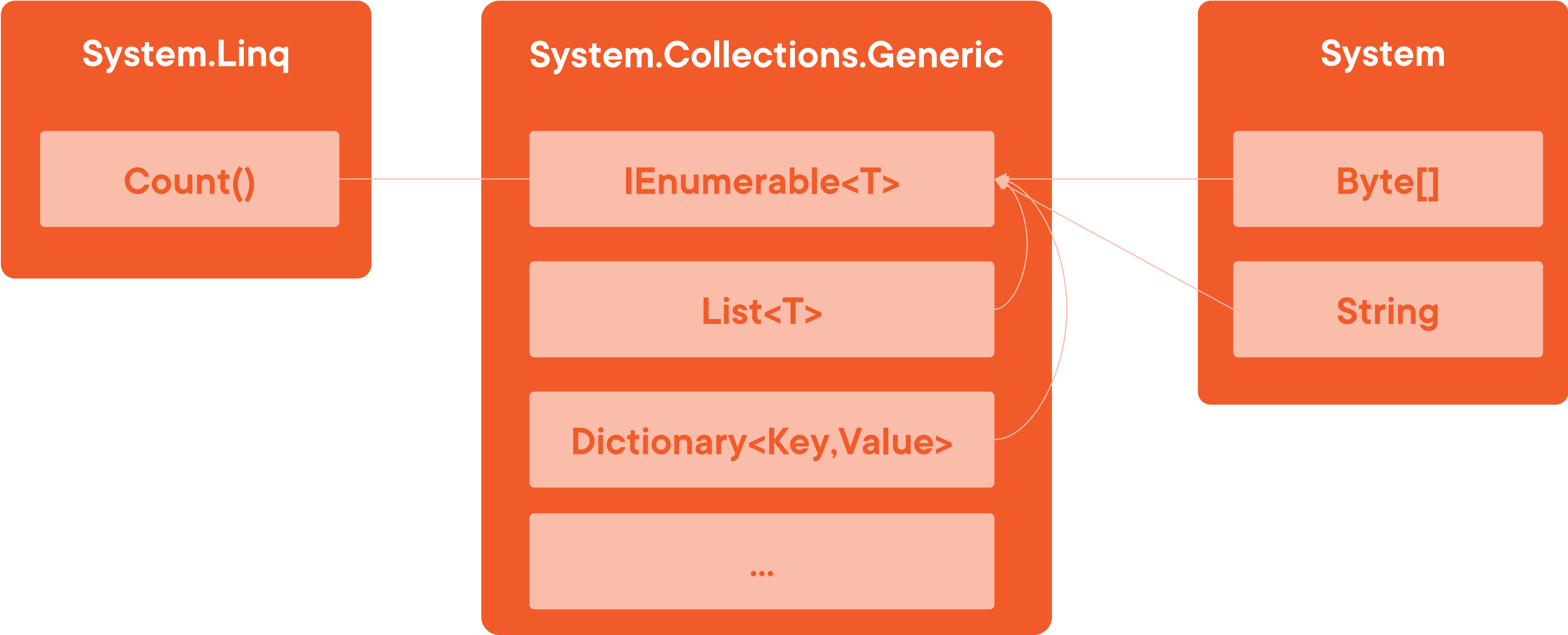
**Count()**

**System.Collections.Generic**

**IEnumerable<T>**

### **Extension method**

Defined in a  
different class, in a  
different assembly



```
public interface IMyInterface
{
    void NewMethod();
}
```

◀ New method defined here

```
public class Class1 : IMyInterface
{
    public void NewMethod() {}
}
```

◀ Every class must implement

```
public class Class2 : IMyInterface
{
    public void NewMethod() {}
}
```

```
public interface IMyInterface
{
    void NewMethod();
}
```

◀ New method defined here

```
abstract class BaseClass : IMyInterface
{
    public void NewMethod() { }
}
```

◀ Abstract base class implements

```
public class Class1 : BaseClass
{
}
```

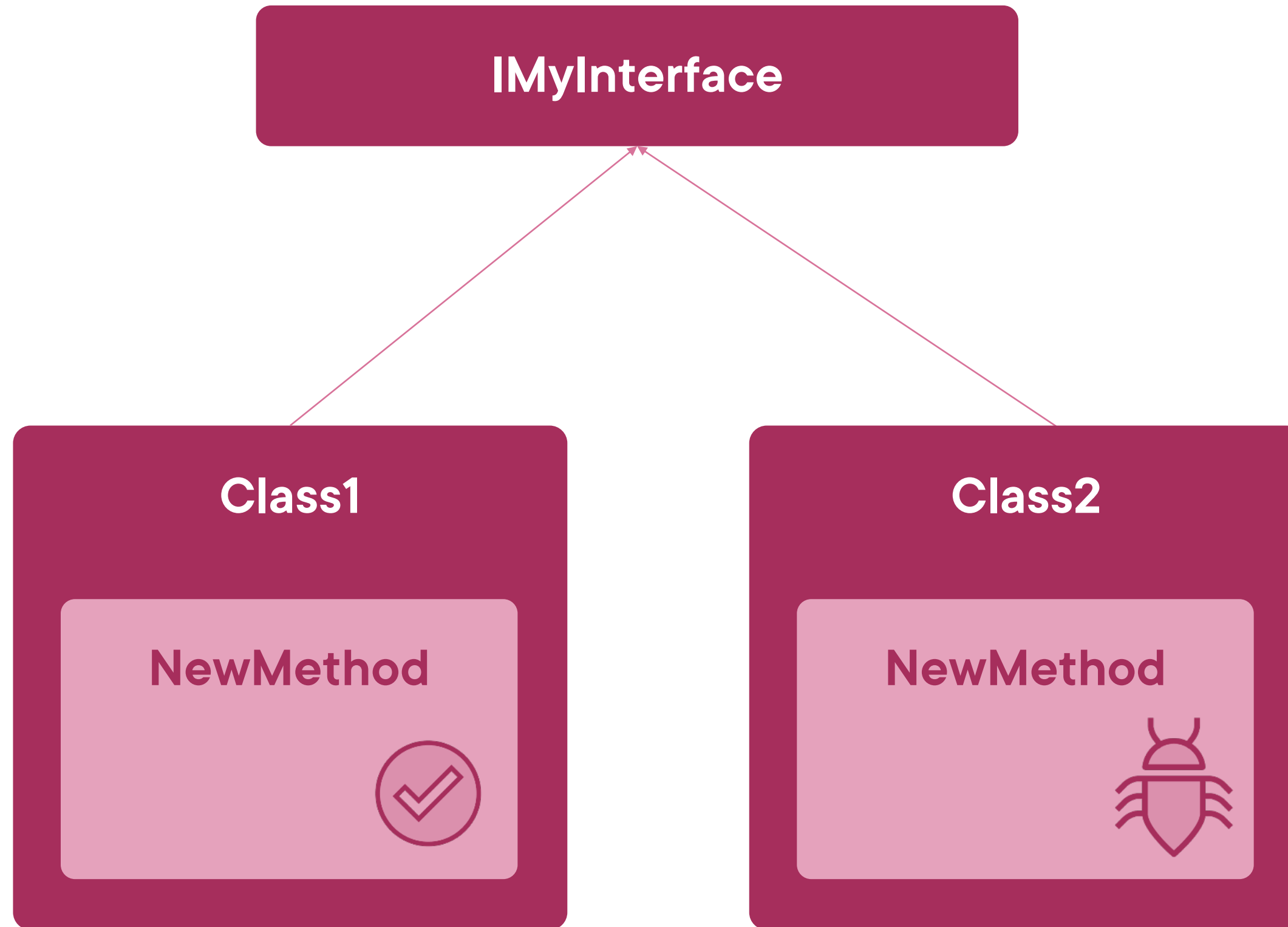
◀ Classes inherit base

D. R. Y.

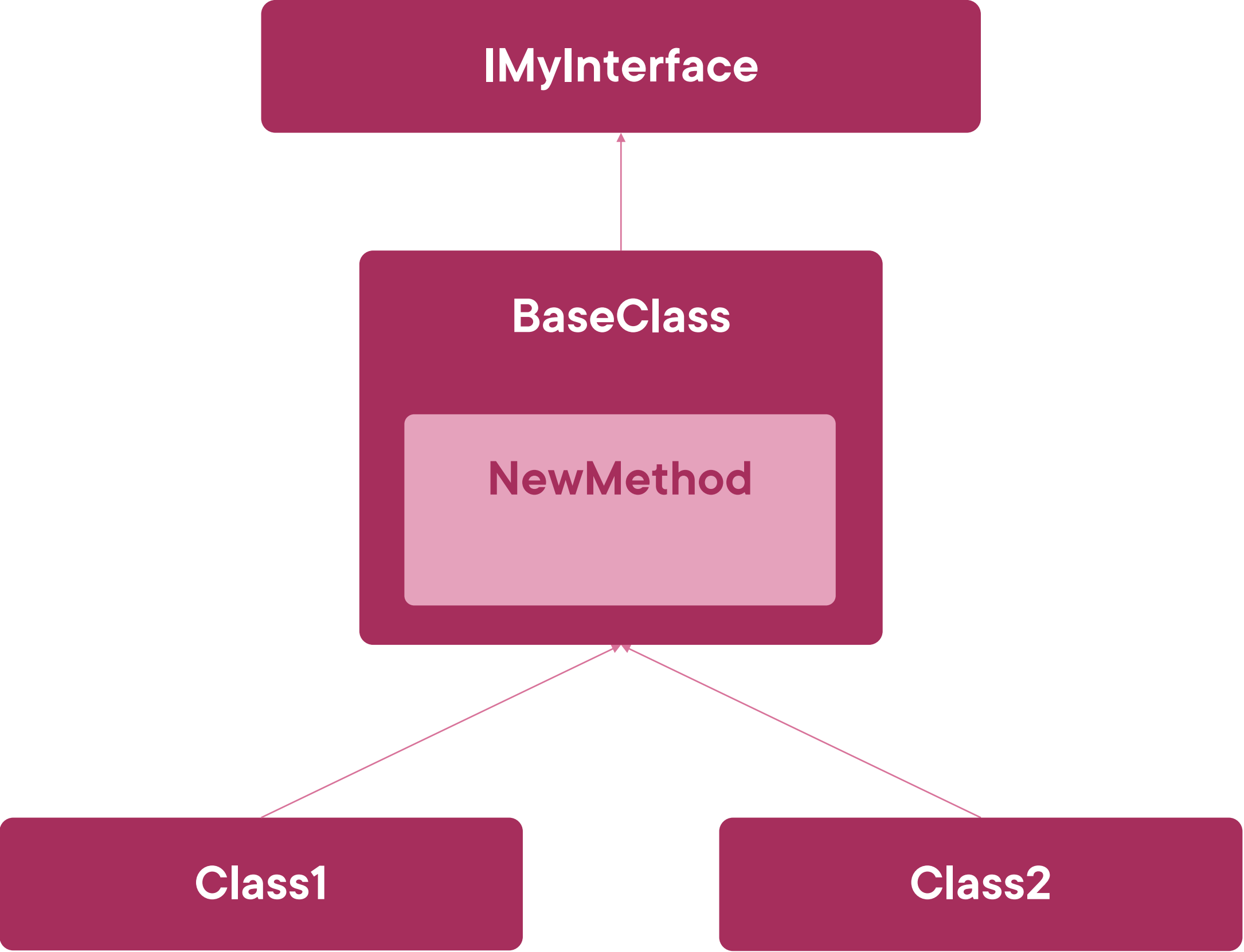
**Don't**

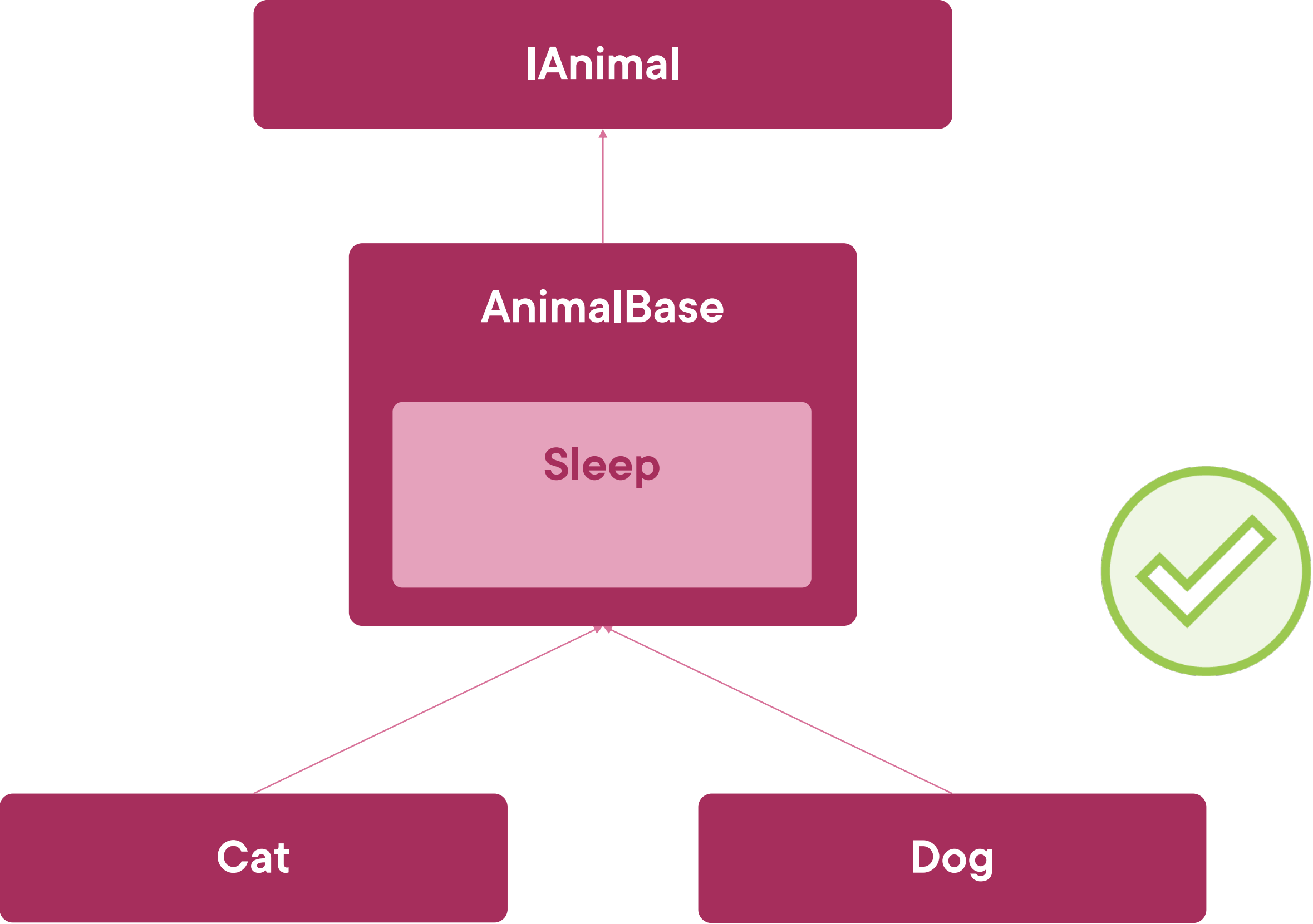
**Repeat**

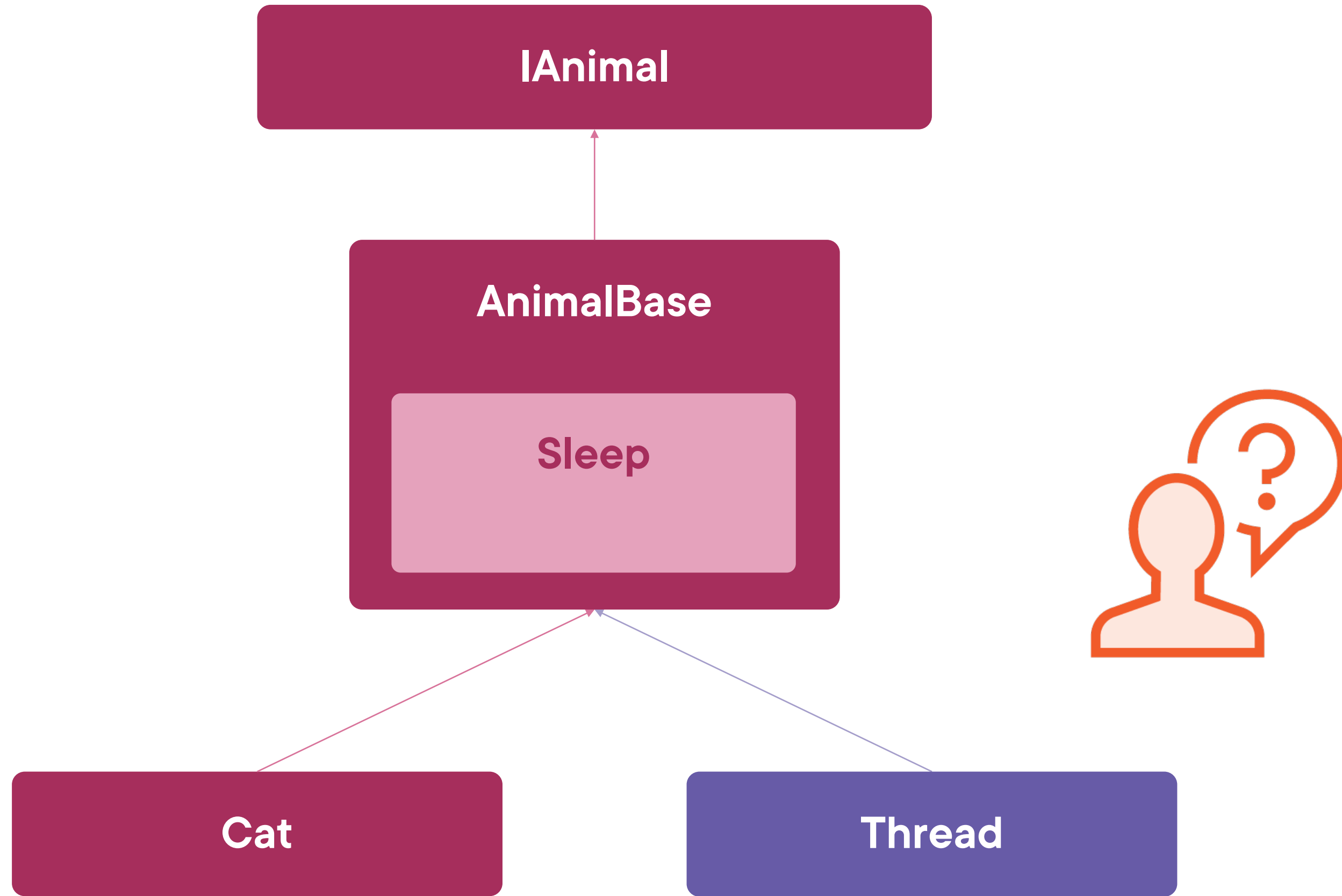
**Yourself**

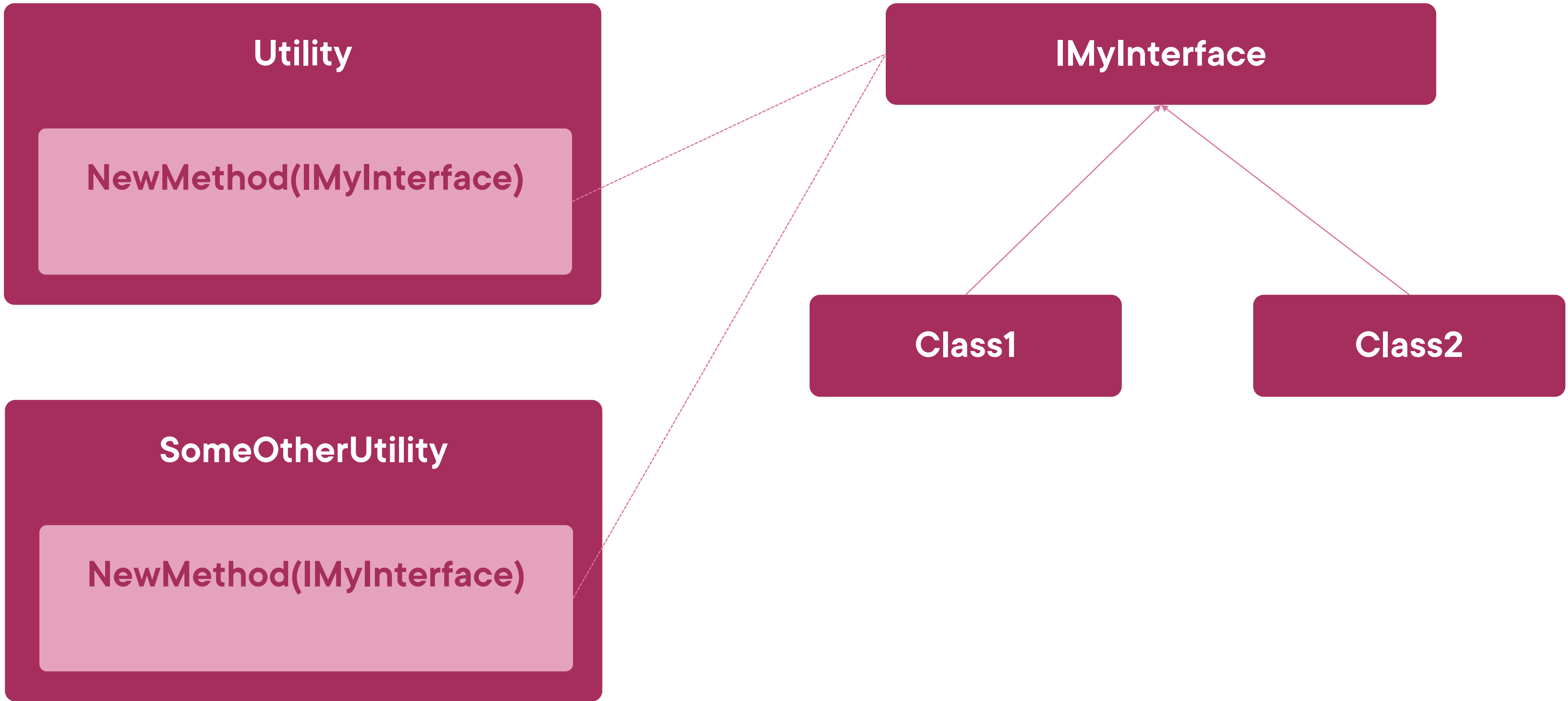


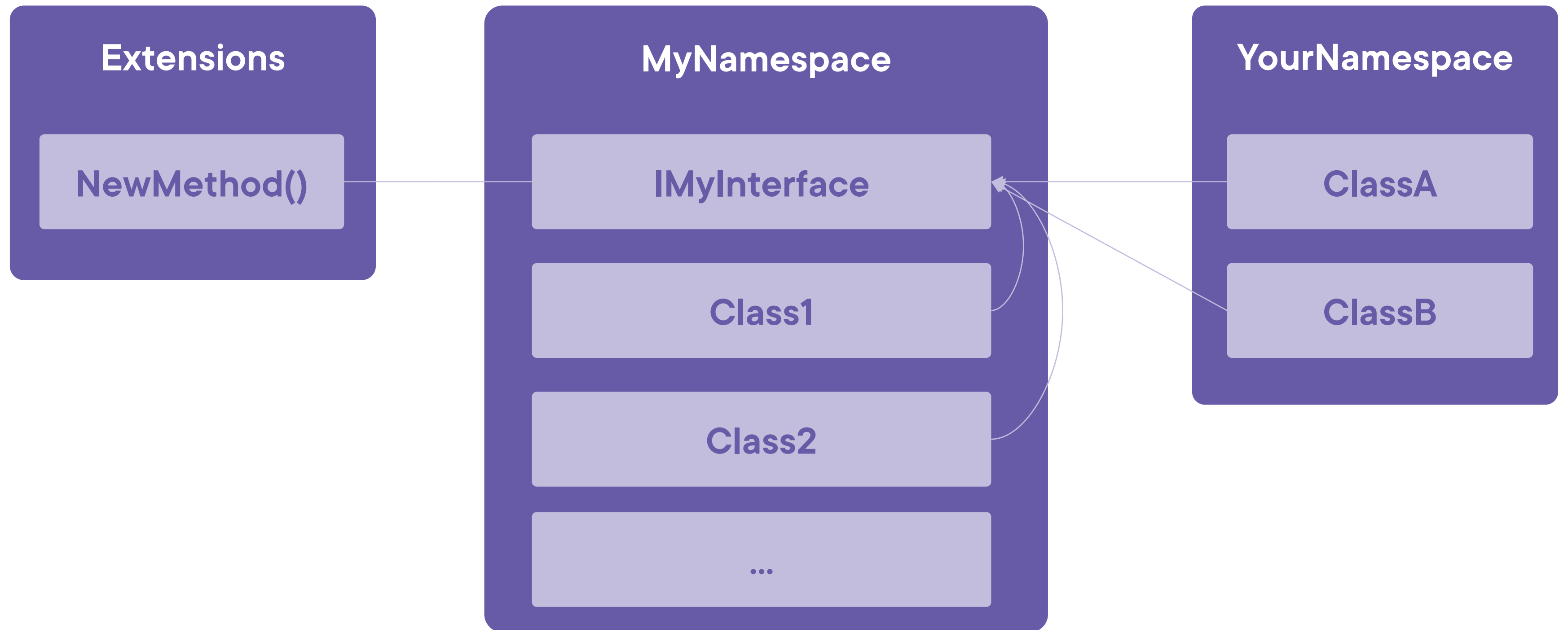












# S. O. L. I. D.

**Single  
Responsibility**

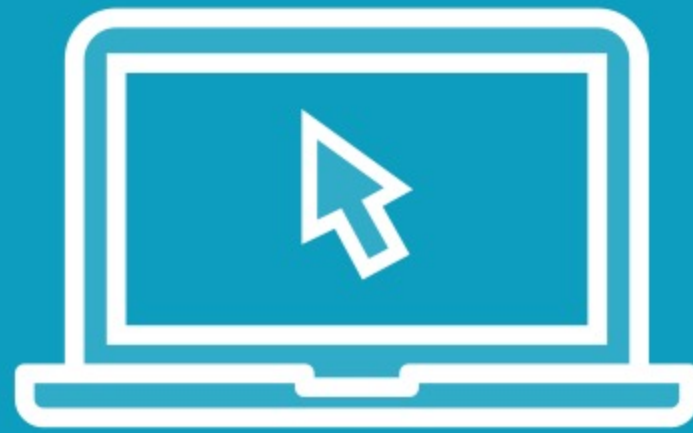
**Open-Closed**

**Liskov  
Substitution**

**Interface  
Segregation**

**Dependency  
Inversion**

# Demo



## Using extension methods

- **Extension methods in the .NET BCL**
- **Extending interfaces and classes**
- **Applying extensions to new classes**

Extension Methods

Visual Studio 2008



Visual Studio Code

JetBrains Rider

C# 3.0



.NET Framework

Mono

.NET Standard

.NET Core



```
var strings = new List<string> { "a", "b", "c" };  
Assert.IsFalse(strings.IsCountEven());
```

IsCountEven() works on IEnumerable

**The method is actually defined in a separate class**

```
using System.Collections.Generic;
using System.Linq;

namespace ExtensionMethods.Tests
{
    public static class EnumerableExtensions
    {
        public static bool IsCountEven<TSource>(this IEnumerable<TSource> target)
        {
            //...
        }
    }
}
```

```
Assert.IsFalse(strings.IsCountEven());
```

```
strings.IsCountEven();
```

```
EnumerableExtensions.  
    IsCountEven(strings);
```

◀ Using the extension method

◀ You write this

◀ The compiler generates this

```
// this code:
```

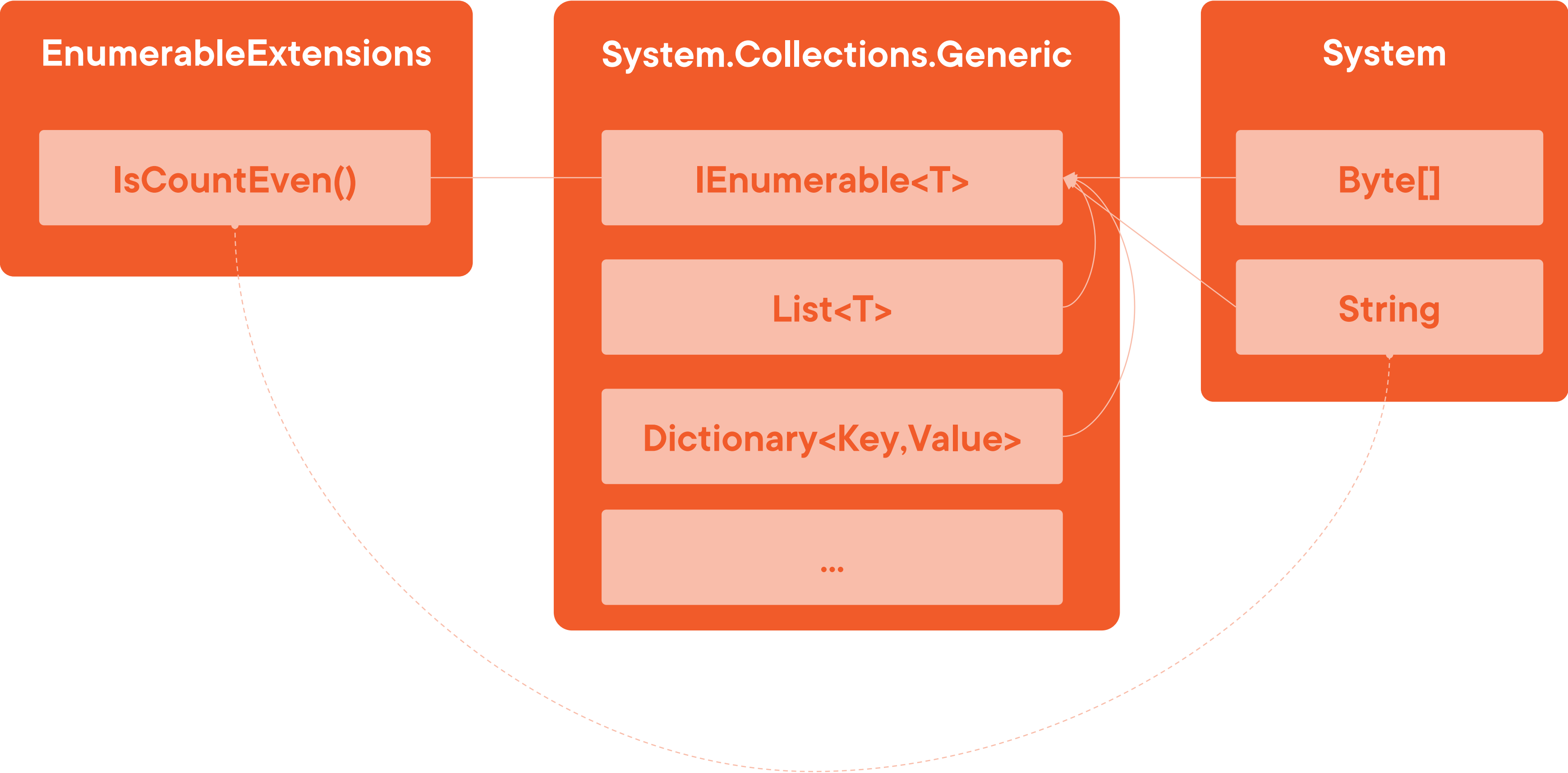
```
strings.IsCountEven();
```

```
// really generates this:
```

```
EnumerableExtensions.IsCountEven(strings);
```

Invocation is on the static class

**The extension method can only use accessible members**



# Demo



## Writing extension methods

- Extending .NET Standard config
- Using extensions to centralize code
- Adding null checks in extensions

# Demo



## Encapsulating business logic


- **Extending .NET configuration builder**
- **Merging multiple config sources**
- **Isolating rules in extension methods**




# Modernizing .NET Framework Apps with Docker


by Elton Stoneman

Docker can help you bring your existing applications into the modern world. This course teaches you how to run full .NET applications in Windows containers, modernize the architecture, and deploy to the cloud.

 Start Course

 Bookmark

 Add to Channel

 Download Course

- Table of contents
- Description**
- Transcript
- Exercise files
- Discussion
- Learning Check
- Recommended

Docker isn't just for greenfield microservices applications, you can take full .NET Framework applications and run them in containers with no code changes. That's a great starting point for modernizing the architecture and moving to the cloud. In this course, Modernizing .NET Framework Apps with Docker, you'll learn how to efficiently run .NET applications and create a more modern architecture utilizing Docker. First, you'll discover how to package and run .NET apps in Docker containers on Windows. Then, you'll explore how to evolve the application architecture by breaking features out into separate containers. Finally, you'll delve into taking your modernized app to production on Azure. By the end of the course, you'll understand how Docker works on Windows and what Docker can do for your existing .NET landscape. Software required: Docker.

Course author



Elton Stoneman

Elton is an 8-time Microsoft MVP, author, trainer and speaker. He spent most of his career as a consultant working in Microsoft technologies, architecting and delivering complex solutions for...

### Course info

Level Intermediate

Rating ★★★★★ (66)

Duration 3h 42m

Released 28 Dec 2017

Share course





```
namespace ExtensionMethods.Library
{
    public static class ConfigurationExtensions
    {
        public static bool IsLoaded(this IConfiguration config)
        {
            return config != null && config.AsEnumerable().Any();
        }
    }
}
```

```
namespace ExtensionMethods.Library
{
    public static class ConfigurationExtensions
    {
        //...
    }
}
```

Namespace defines scope

**Consumers reference the library and namespace**

```
namespace Microsoft.Extensions.Configuration
{
    public static class ConfigurationExtensions
    {
        //...
    }
}
```

Namespace affects discoverability

**Use the target type's namespace or your own**

```
public static class ConfigurationExtensions
{
    public static bool IsLoaded(this IConfiguration config)
    {
        //...
    }
}
```

Accessibility rules apply

**Types must be accessible to the extension method library**

```
public static bool IsLoaded(this IConfiguration config)
{
    return config.AsEnumerable().Any();
}
```

Accessibility rules apply

**Members of the target type must be accessible**

```
public static bool IsLoaded(this IConfiguration config)
{
    return config.AsEnumerable().Any();
}
```

Type methods take precedence

**Extension methods with matching signatures are hidden**

```
[Test]
public void IsLoaded()
{
    IConfiguration config = null;
    Assert.IsFalse(config.IsLoaded());
}
```

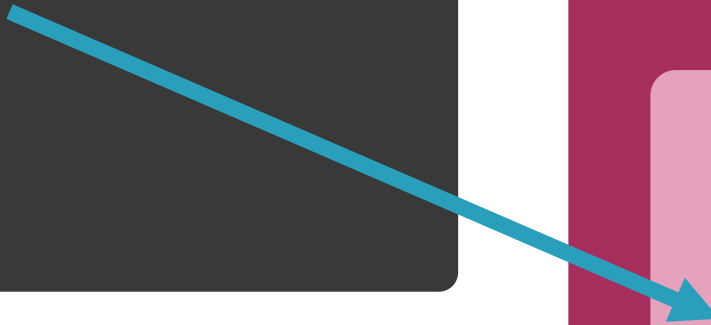
Microsoft.Extensions.Configuration

IConfiguration

ExtensionMethods.Library

ConfigurationExtensions

IsLoaded()



```
[Test]
public void IsLoaded()
{
    IConfiguration config = null;
    Assert.IsFalse(config.IsLoaded());
}
```

Microsoft.Extensions.Configuration

IConfiguration

IsLoaded()

ExtensionMethods.Library

ConfigurationExtensions

IsLoaded()



# Demo



## Limitations of extension methods

- Working with type accessibility
- Using accessible members
- Type methods hiding extensions

**ExtensionMethods.Library**

**ConfigurationExtensions**

**IsLoaded()**

**Microsoft.Extensions.Configuration**

**IConfiguration**

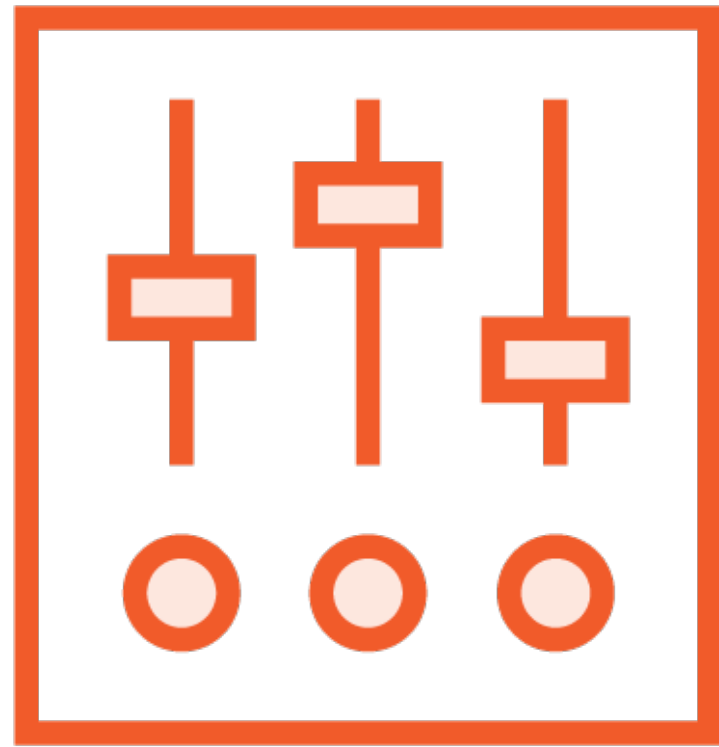


```
using ExtensionMethods.Library;  
  
//...  
  
IConfiguration config = null;  
Assert.IsFalse(config.IsLoaded());
```

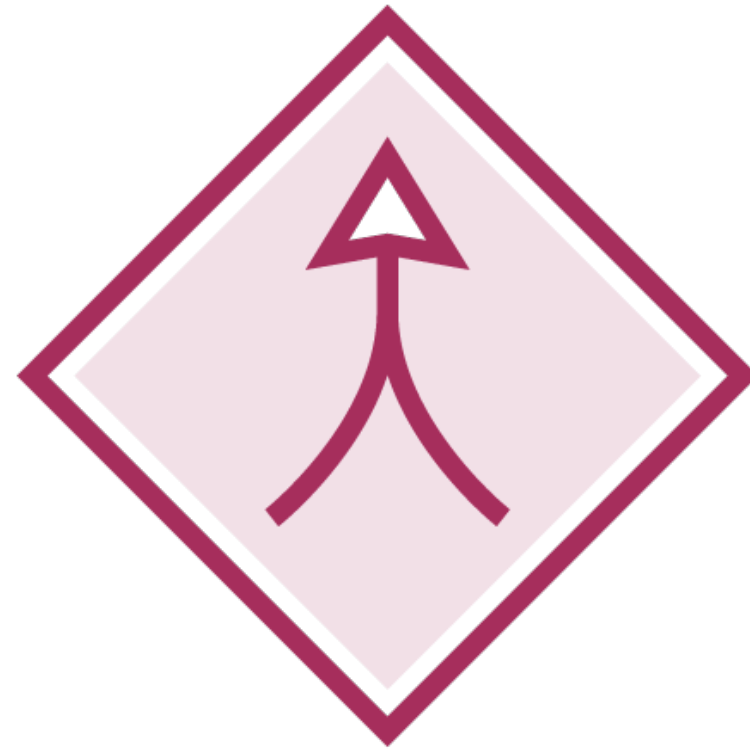
Using extension methods

**Reference the library and the namespace**

```
namespace ExtensionMethods.Library
{
    public static class ConfigurationExtensions
    {
        public static bool IsLoaded(this IConfiguration config)
        {
            return config != null && config.AsEnumerable().Any();
        }
    }
}
```



**Custom behavior**  
**Same usage patterns**



**No subclasses**  
**No new interfaces**



**Centralized libraries**  
**Clean dependencies**

Up Next:

Best Practices for Using Extension Methods

---