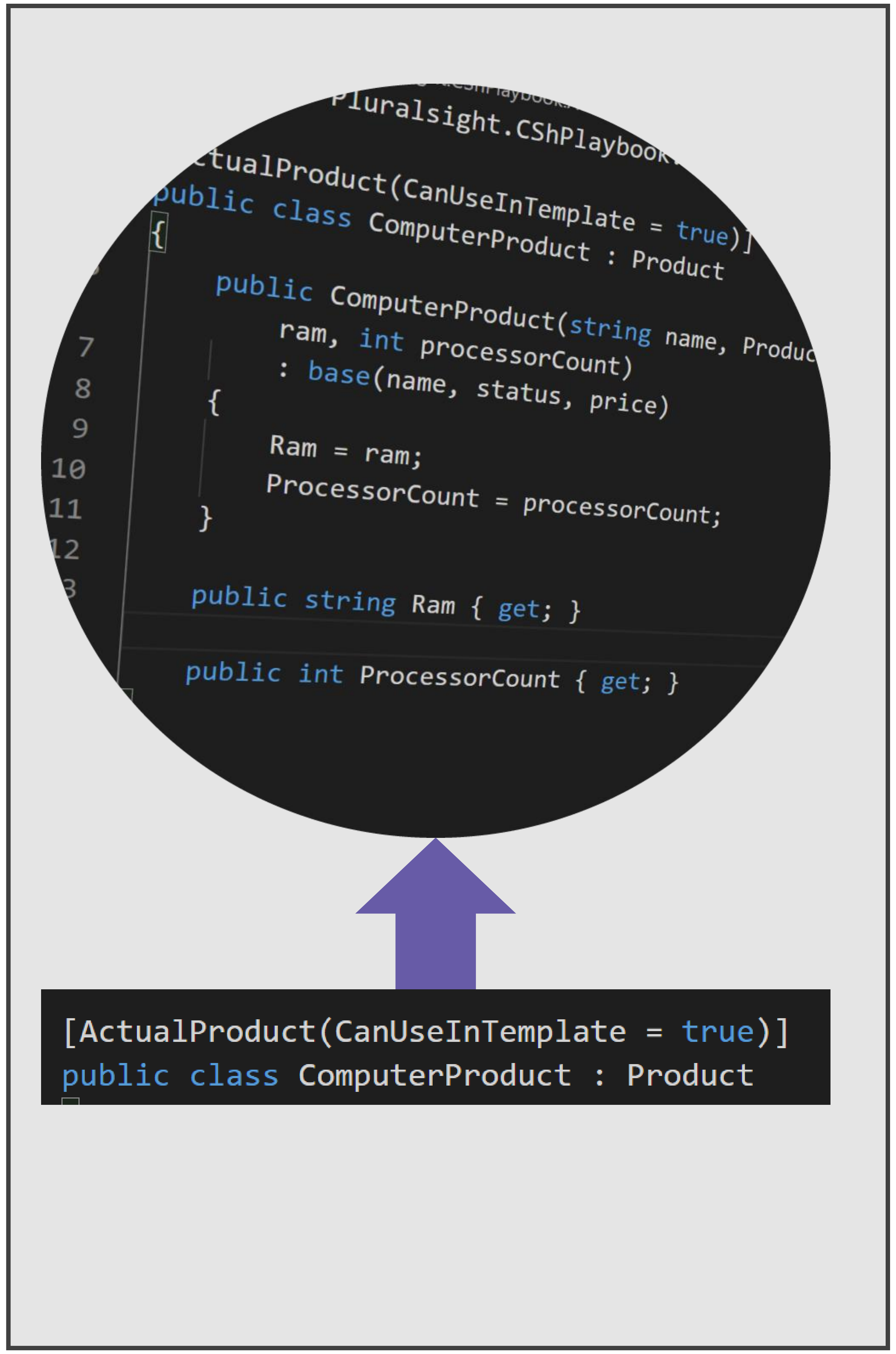
Attributes and Reflection

Simon Robinson
Software Developer

@TechieSimon www.SimonRobinson.com



```
Pluralsight.CShPlaybook
tualProduct(CanUseInTemplate = true)]
public class ComputerProduct : Product
{
    public ComputerProduct(string name, Produc
        ram, int processorCount)
        : base(name, status, price)
    {
        Ram = ram;
        ProcessorCount = processorCount;
    }

    public string Ram { get; }

    public int ProcessorCount { get; }
```

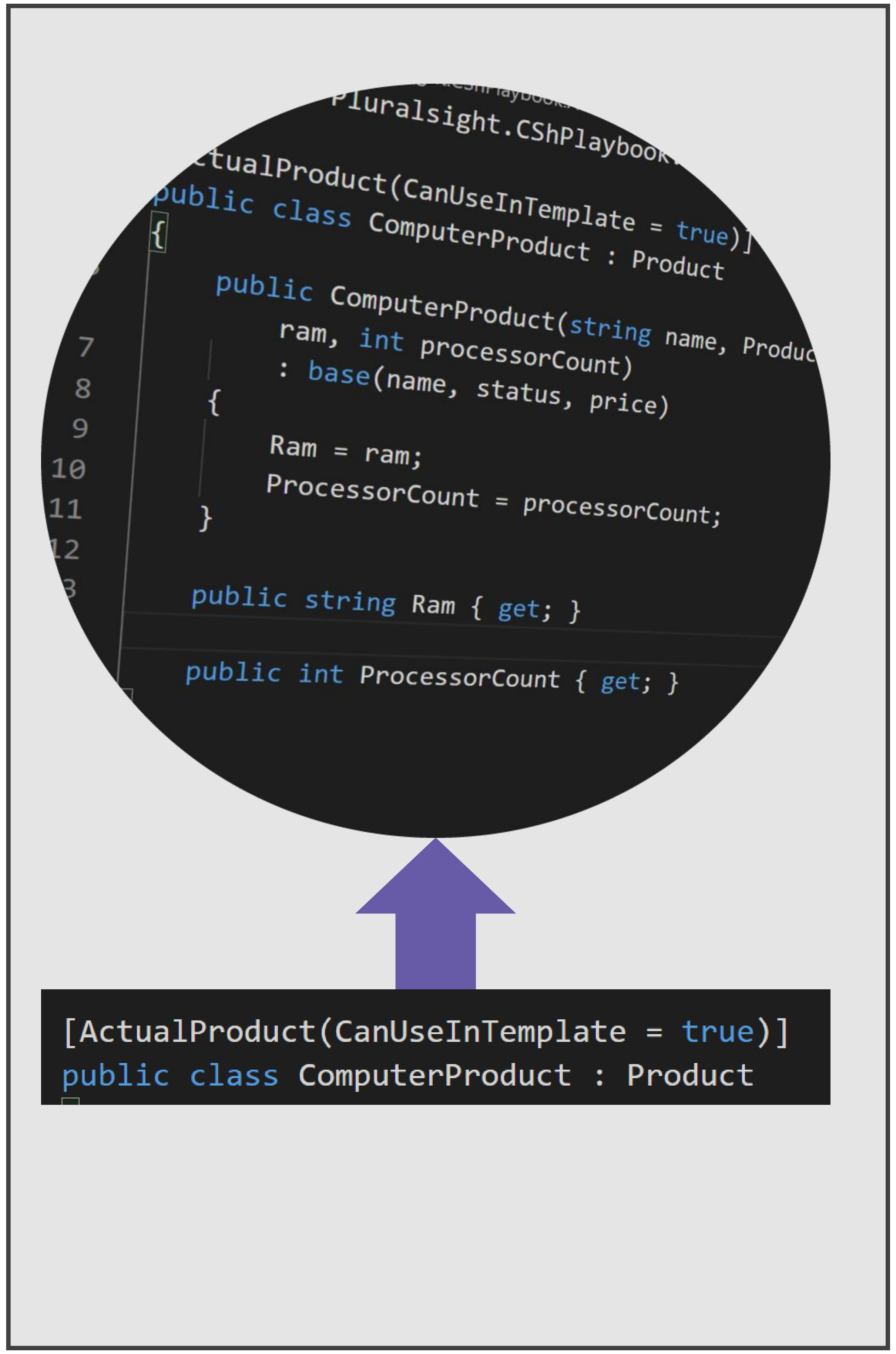
```
[ActualProduct(CanUseInTemplate = true)]
public class ComputerProduct : Product
```

Reflection

- Find out about types in the code
- Use cases:
  - Persisting or displaying types
  - Diagnostic tools

Attributes

- Attach metadata to code
- Can be read using reflection


```
Pluralsight.CShPlaybook
[ActualProduct(CanUseInTemplate = true)]
public class ComputerProduct : Product
{
    public ComputerProduct(string name, Produc
        ram, int processorCount)
        : base(name, status, price)
    {
        Ram = ram;
        ProcessorCount = processorCount;
    }

    public string Ram { get; }

    public int ProcessorCount { get; }
```

```
[ActualProduct(CanUseInTemplate = true)]
public class ComputerProduct : Product
```

Demo app to read product data

- Using reflection
- Will give flavour of how attributes and reflection are used

Overview

Solutions using attributes and reflection

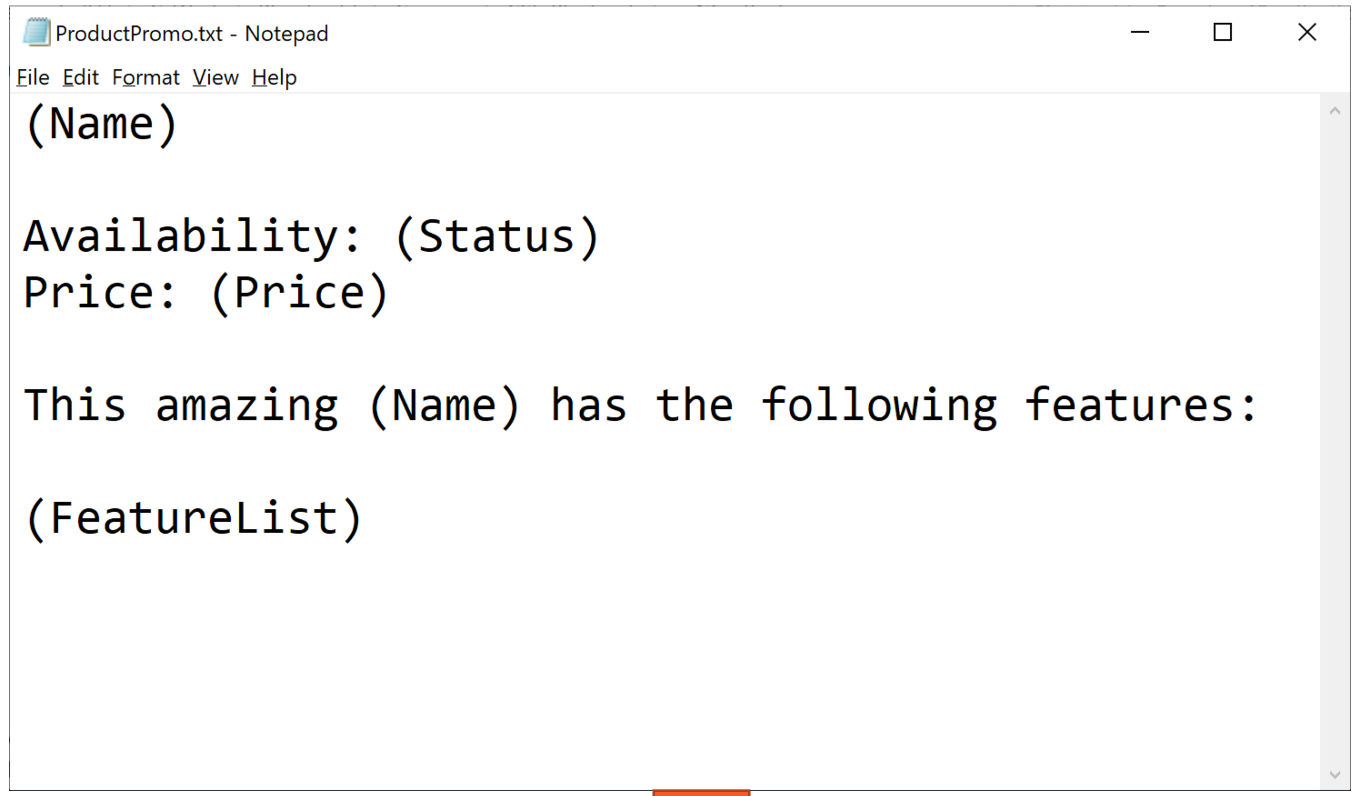
- Marking a method as obsolete
- Identifying attributes on a type
- Creating friendly text for enums
- Finding out what properties an instance has
  - And writing out their values
- Finding out if a type is immutable

The Demo

Displays product data

Text template file controls what is displayed

The code substitutes actual values for the placeholders

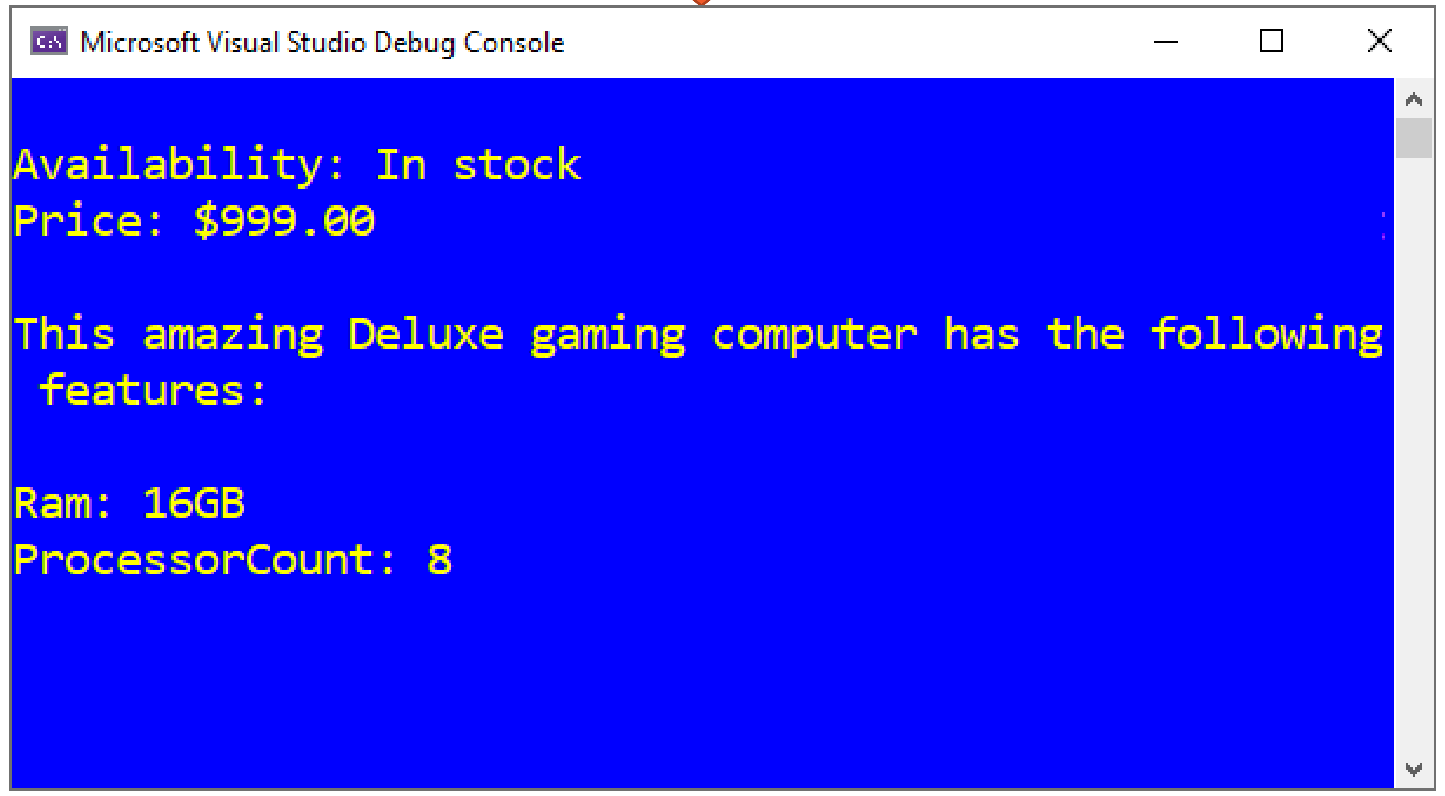


```
(Name)

Availability: (Status)
Price: (Price)

This amazing (Name) has the following features:

(FeatureList)
```



```
Availability: In stock
Price: $999.00

This amazing Deluxe gaming computer has the following
 features:

Ram: 16GB
ProcessorCount: 8
```

Demo

Set up the demo

- Show how it processes the template

Marking a Method as Obsolete

This is useful if you have replaced a method in a library, but need to leave the old method to support legacy code

Demo

Replace text template format

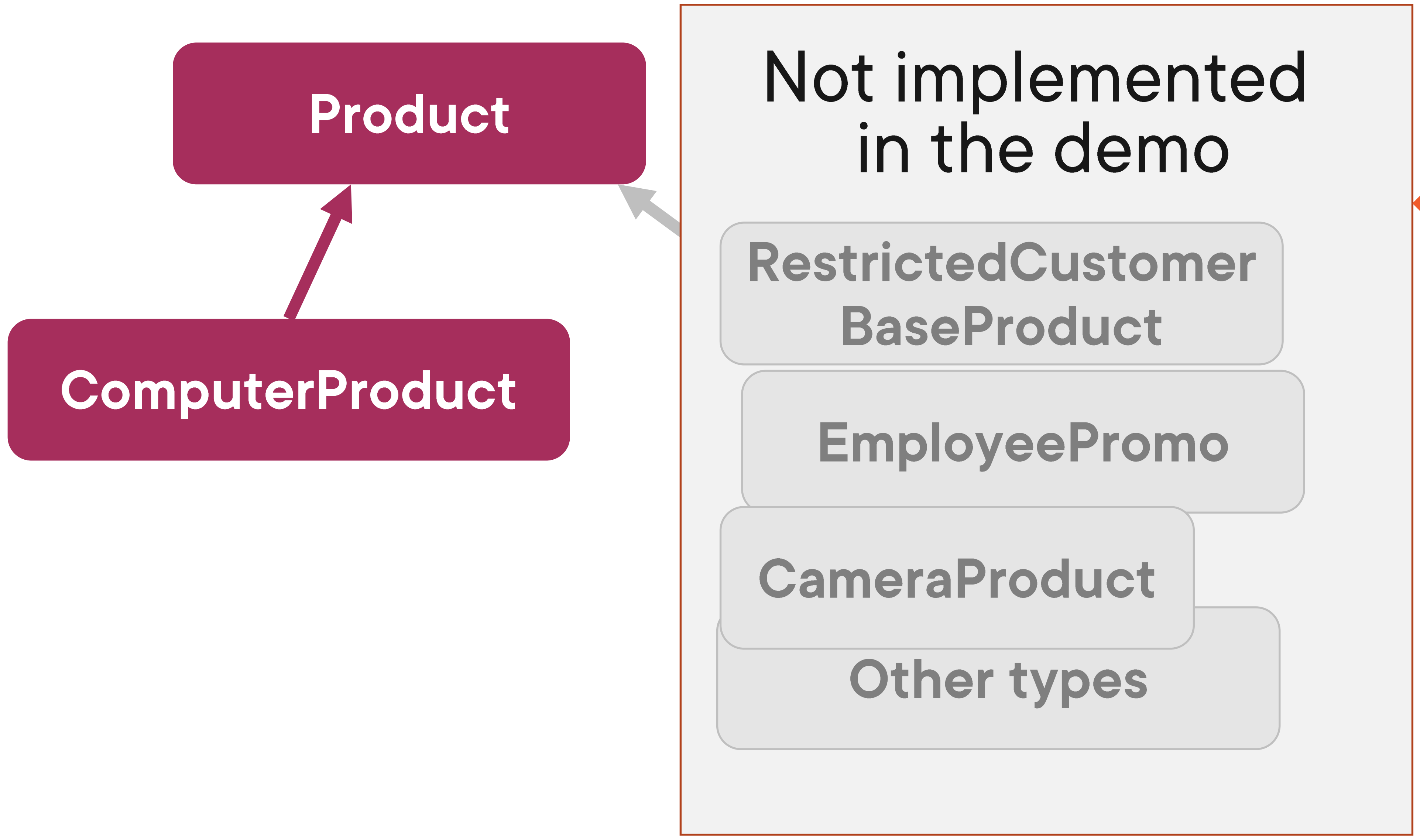
- Provide new `GenerateText()` method
- Use `Obsolete` attribute to mark the old method

Using Reflection to Check for Attributes

The Problem to Solve

```
public string GenerateTextV2(Product product)
{
```

This will accept anything that inherits from Product



Not all these types should be permitted when generating sales text

The task:
Restrict `GenerateTextV2()` to only accept types marked as suitable

Demo

Use an attribute to mark suitable product types

- Use reflection to test for the attribute

Displaying Friendly Text for Enum Values

Demo

Create an attribute to add human-friendly text to enum values

- Use reflection to extract that text

Getting the Property Values of an Instance

Previously:

Type

Used reflection
to obtain
compile-time information
from a type

Next:

Instance

Will use reflection
to obtain
run-time information
from an instance

Demo

Implement the feature list in the template

- Use reflection to read whatever property values are available

Identifying Whether a Class is Immutable

Demo

Have text generator say whether a product type is immutable

- Use reflection to determine this

Summary

Attributes

- Add metadata to code elements
- Some MS attributes are recognised by the compiler
  - `ObsoleteAttribute`
- Define your own attributes
  - Inherit from `System.Attribute`
  - Use `AttributeUsageAttribute` to specify how your attribute should be used

Summary

Reflection

- Lets you inspect code
- `System.Type` is the usual entry point
  - `GetProperties()`, `GetFields()`, etc.
  - `GetCustomAttributes()`
- Use case examples:
  - Creating friendly text for enums
  - Finding out if a type is immutable
  - Finding property values, even if you don't know what properties exist