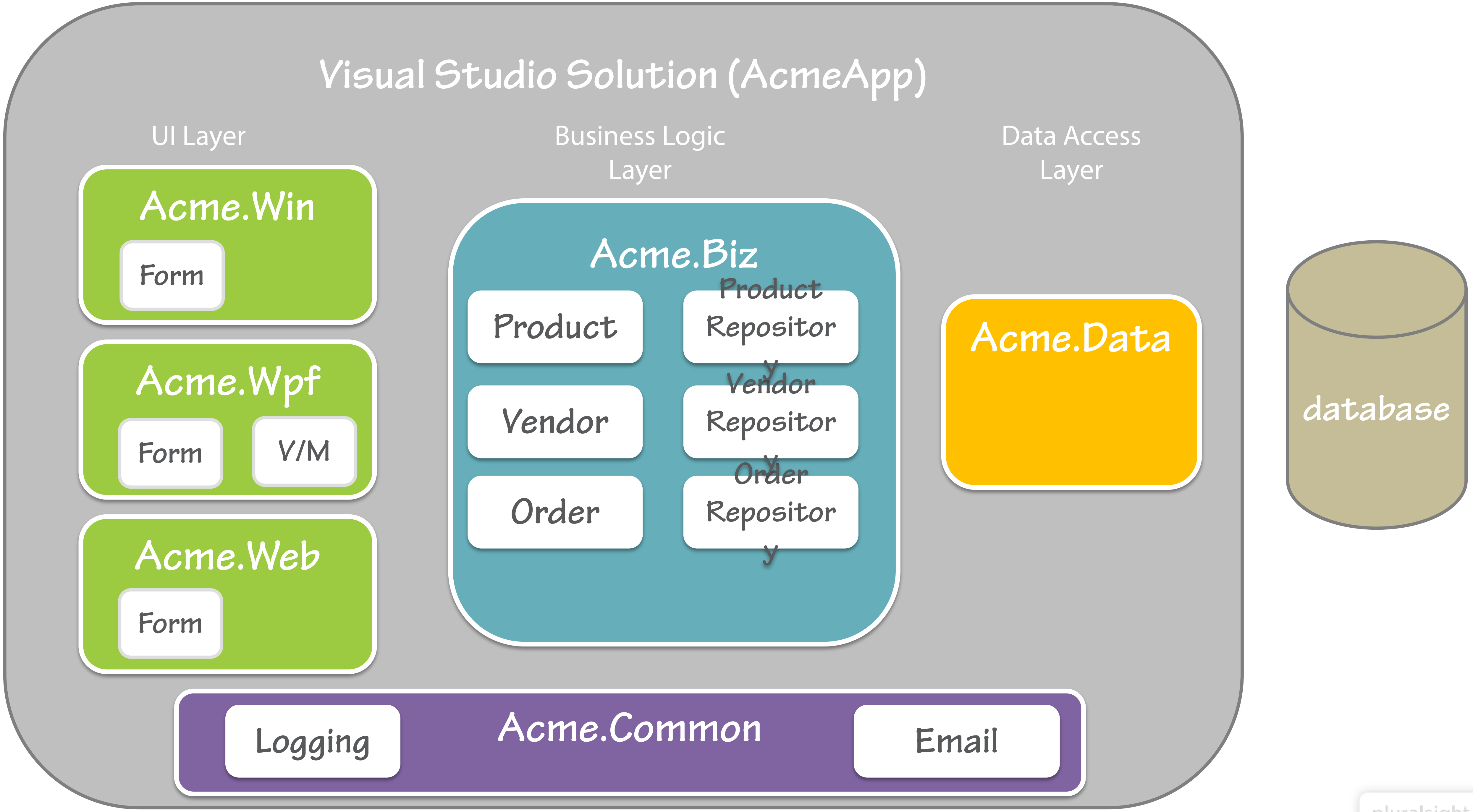
Accessing and Using Classes

Deborah Kurata

@deborahkurata | blogs.msmvps.com/deborahk/

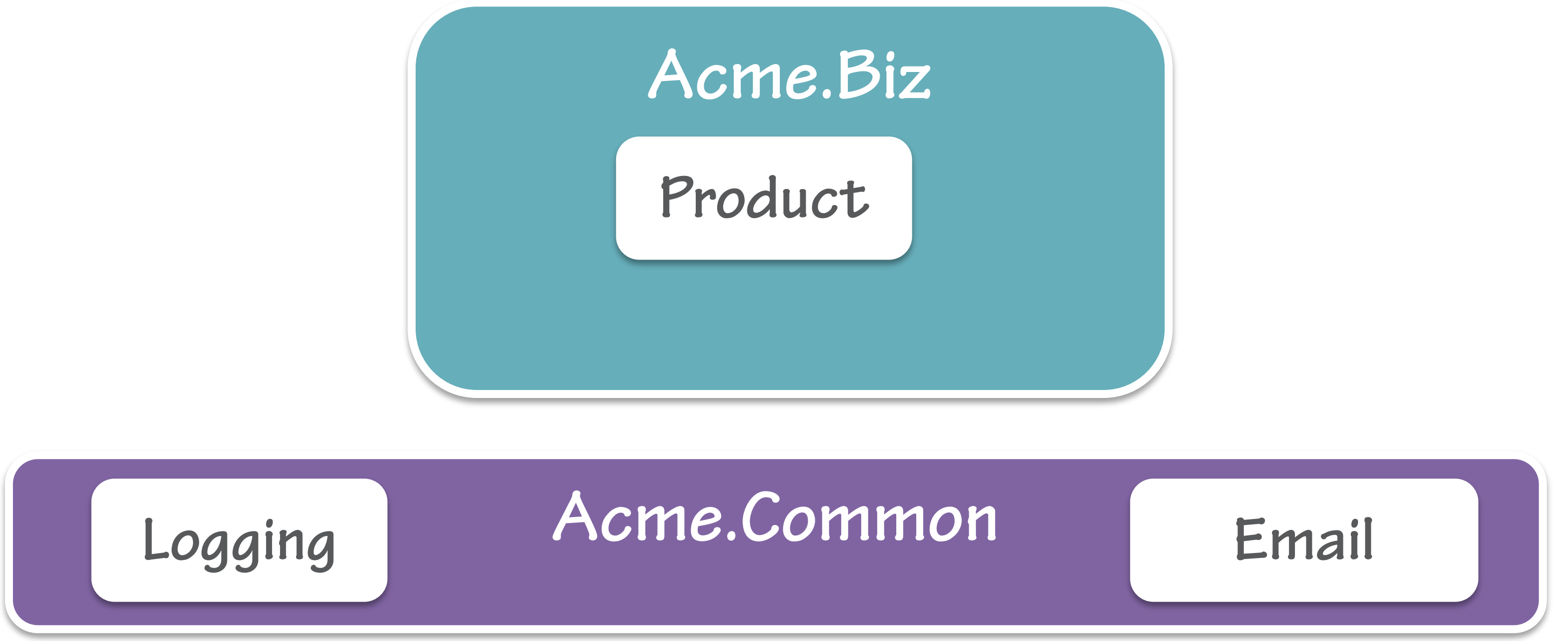

Visual Studio Solution (AcmeApp)
UI Layer
Business Logic Layer
Data Access Layer
Acme.Win
Form
Acme.Wpf
Form
V/M
Acme.Web
Form
Acme.Biz
Product
Product Repository
Vendor
Vendor Repository
Order
Order Repository
Acme.Data
database
Logging
Acme.Common
Email

Module Overview

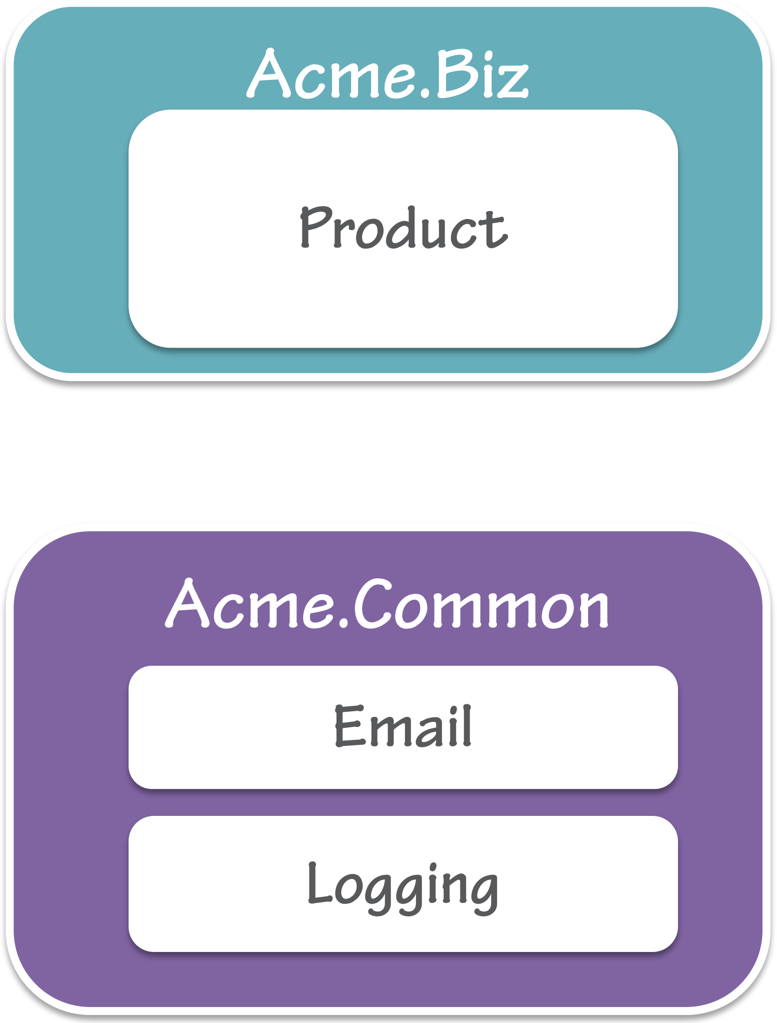
- References and Using
- Using a Class
- Object Initialization
- Instantiating Related Objects
- Null Checking
- FAQ

References and Using

- Define a reference
  - Identifies the component containing the class
- Use a `using` directive
  - To use the class without the fully qualified namespace



References and Using Best Practices



Do:

Take care when defining references

References must be one way

Take advantage of the `using` directive

Avoid:

Excessive use of the `using static` directive

Using a Class

Accessing class members

Terminology

Accessing Class Members

Non-static Class

```
var currentProduct = new Product();
var result = currentProduct.SayHello();
```

Static Class

```
var result = LoggingService.LogAction("");
```

Object vs. Class

Represents one specific thing
Example: Hammer or Saw

Defines one thing created from that template

Created at runtime with the **`new`** keyword

Represents things of the same type
Example: Product

Defines the template specifying the data and processing associated with all things of that type

Created at development time with code

Object Initialization

- Setting properties
- Parameterized constructor
- Object initializers

Object Initialization Best Practices

```
var currentProduct = new Product();
currentProduct.ProductName = "Saw";
currentProduct.ProductId = 1;
currentProduct.Description =
       "15-inch steel blade hand saw";
```

Setting Properties

When populating from database values

When modifying properties

```
var currentProduct = new Product(1, "Saw",
          "15-inch steel blade hand saw");
```

Parameterized Constructor

When setting the basic set of properties

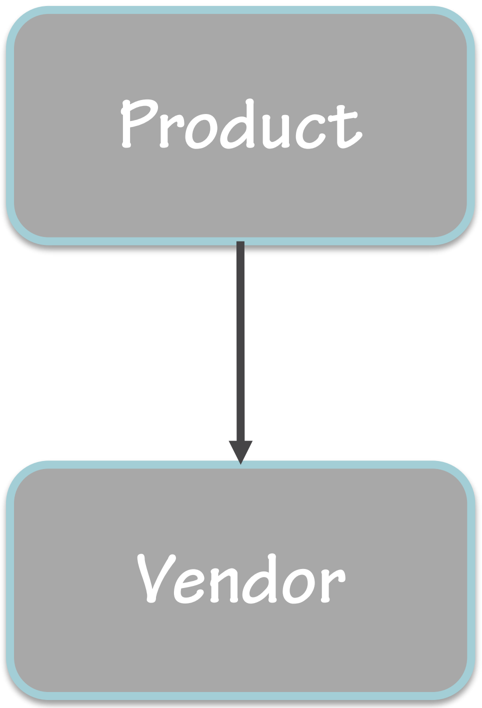
```
var currentProduct = new Product
{
    ProductId = 1,
    ProductName = "Saw",
    Description = "15-inch steel blade hand saw"
};
```

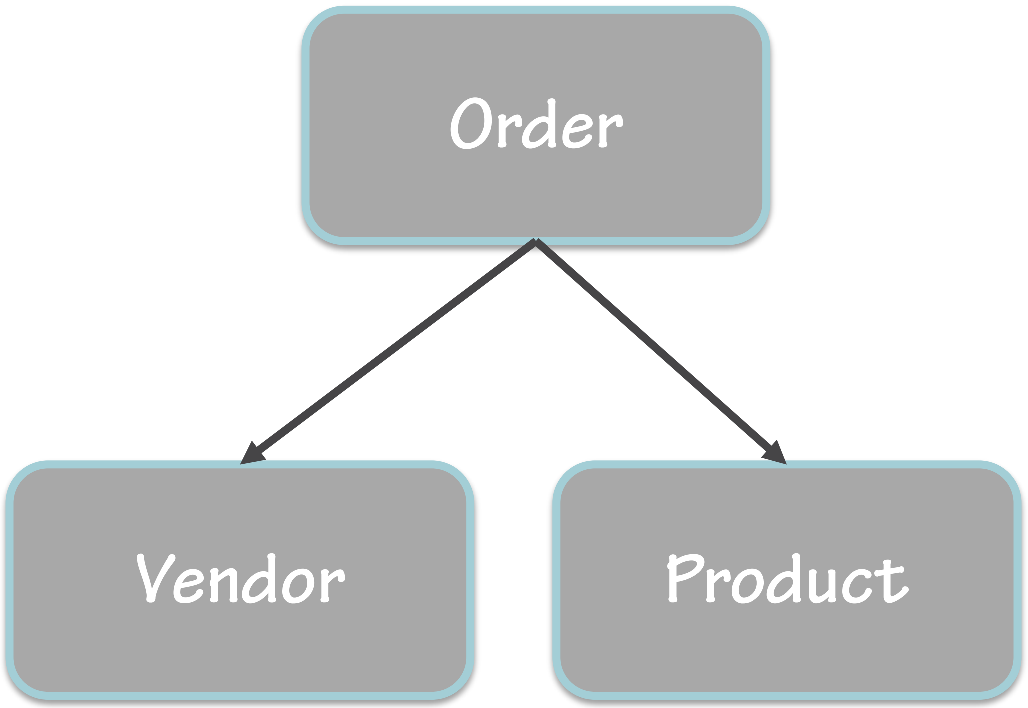
Object Initializers

When readability is important

When initializing a subset or superset of properties

Instantiating Related Objects





Usage Scenarios

- One method
- Always
- Sometimes

Related Object Initialization

```
public string SayHello()
{
    var vendor = new Vendor();
    var vendorGreeting = vendor.SayHello();
    ...
}
```

One method

Initialize in the method that needs it

```
private Vendor productVendor;
public Vendor ProductVendor
{
    get { return productVendor; }
    set { productVendor = value; }
}
public Product()
{
    this.ProductVendor = new Vendor();
}
```

Always

Define a property

Initialize in the constructor

Related Object Initialization

```
private Vendor productVendor;
public Vendor ProductVendor
{
    get
    {
        if (productVendor == null)
        {
            productVendor = new Vendor();
        }
        return productVendor;
    }
    set { productVendor = value; }
}
```

Sometimes

Define a property

Initialize in the property getter

"Lazy Loading"

Null Checking

Object variable is local variable

```
public string SayHello()
{
    var vendor = new Vendor();
    var vendorGreeting = vendor.SayHello();
    ...
}
```

Object variable is a backing field with a property

```
private Vendor productVendor;
public Vendor ProductVendor
{
    get { return productVendor; }
    set { productVendor = value; }
}
```

Null Checking: Classic

```
var companyName = currentProduct.ProductVendor.CompanyName;
```

Null Checking: Null-Conditional Operators

```
var companyName = currentProduct?.ProductVendor?.CompanyName;
```

- ?. Is the null-conditional operator
  - Called the "Elvis operator"
- If the variable on the left side is null, the expression is null
- If the variable on the left side is not null, then we continue with the dot.
- "If null then null; if not then dot" - Mads Torgersen, C# Language PM

Frequently Asked Questions

- What is the difference between an object and a class?
  - A class is a template that specifies the data and operations for an entity
  - An object is an instance of that class created at runtime using the new keyword
- What is lazy loading and when would you use it?
  - Instantiating related objects when they are needed and not before
  - This often involves creating the instance in the property getter for the related object

This Module Covered

- References and Using
- Using a Class
- Object Initialization
- Instantiating Related Objects
- Null Checking