Building Entity Classes

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/

Identified Classes

Customer

- Name
- Email address
- Home address
- Work address
- Validate()
- Retrieve()
- Save()

Product

- Product name
- Description
- Current price
- Validate()
- Retrieve()
- Save()

Order

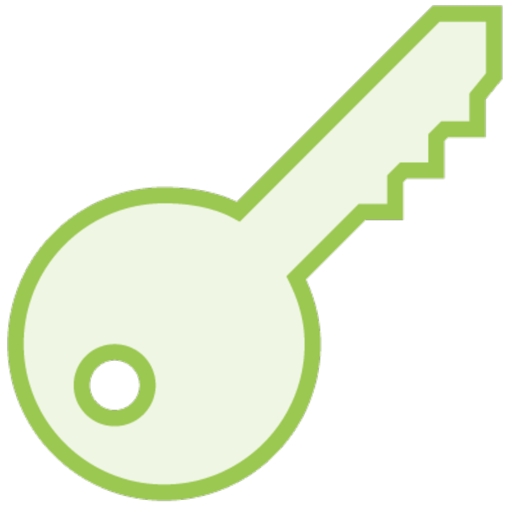
- Customer
- Order date
- Shipping address
- Order items
- Validate()
- Retrieve()
- Save()

Order Item

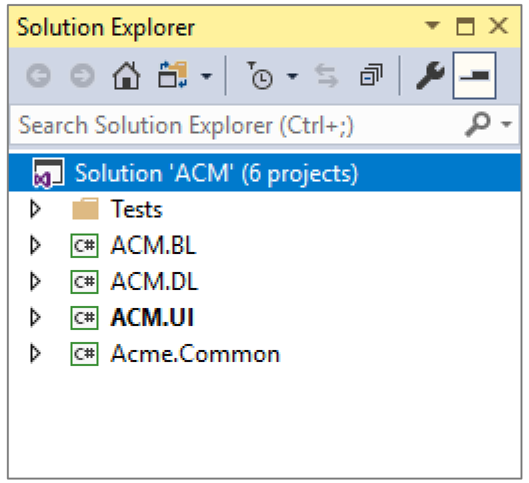
- Product
- Quantity
- Purchase price
- Validate()
- Retrieve()
- Save()

Layering the Application

Build with a layered structure



Layering is key to a good application structure



Solution 'ACM' (6 projects)
Tests
ACM.BL
ACM.DL
ACM.UI
Acme.Common

Solution -> application

Project -> layer

Common Application Layers

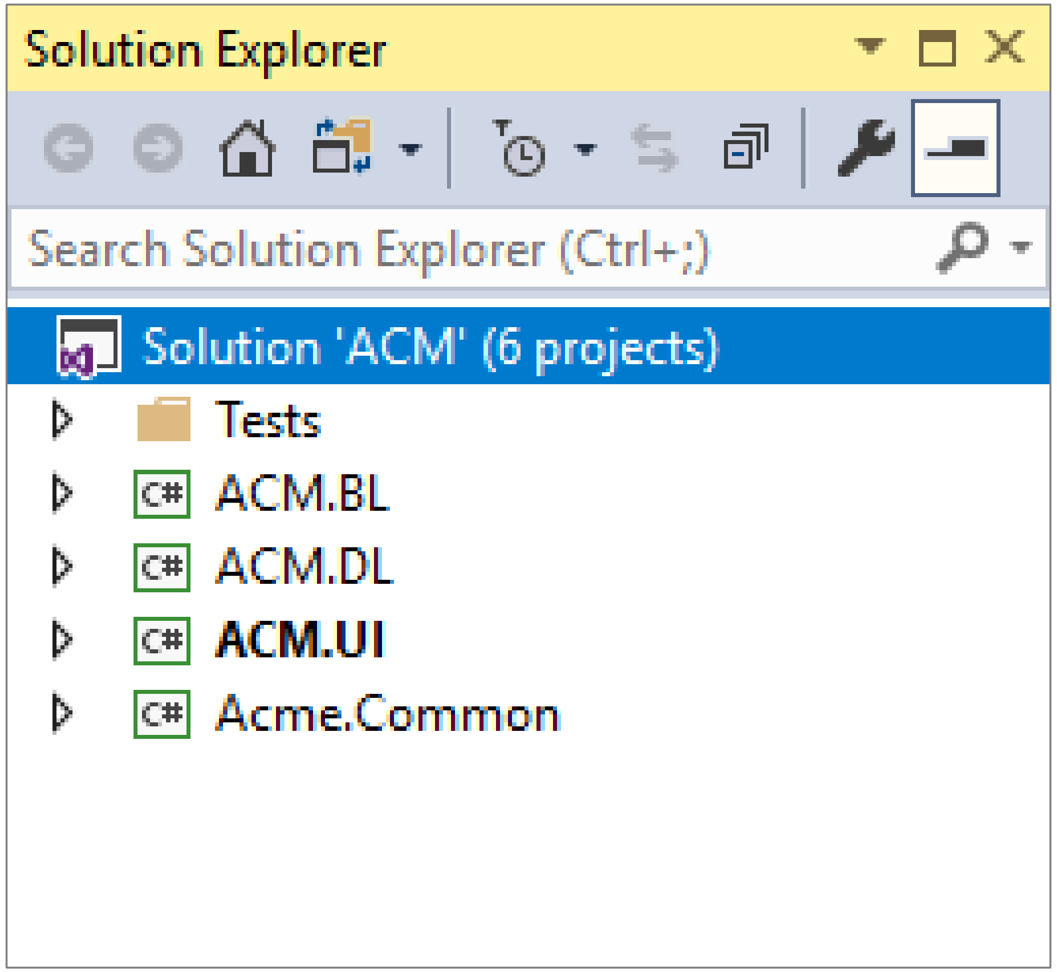
User interface layer

Business logic layer

Data access layer

Common code

Visual Studio Solution



Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'ACM' (6 projects)
- Tests
- ACM.BL
- ACM.DL
- **ACM.UI**
- Acme.Common

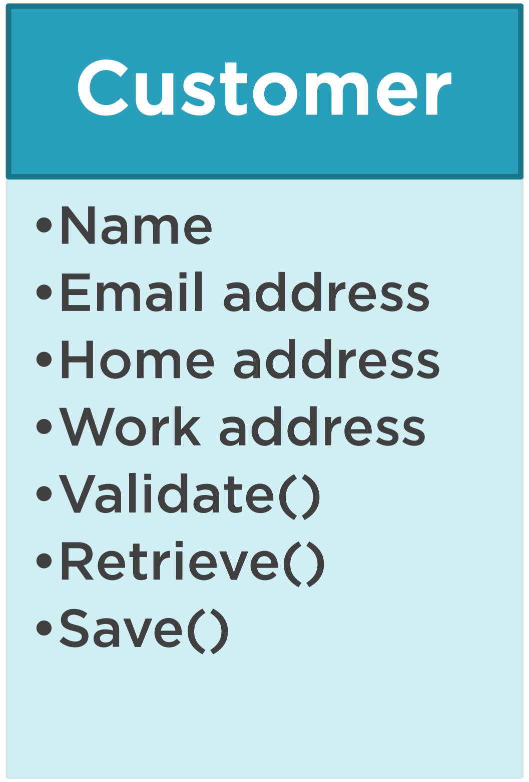
Demo

Building the business logic component

Demo

Building a class

Adding properties



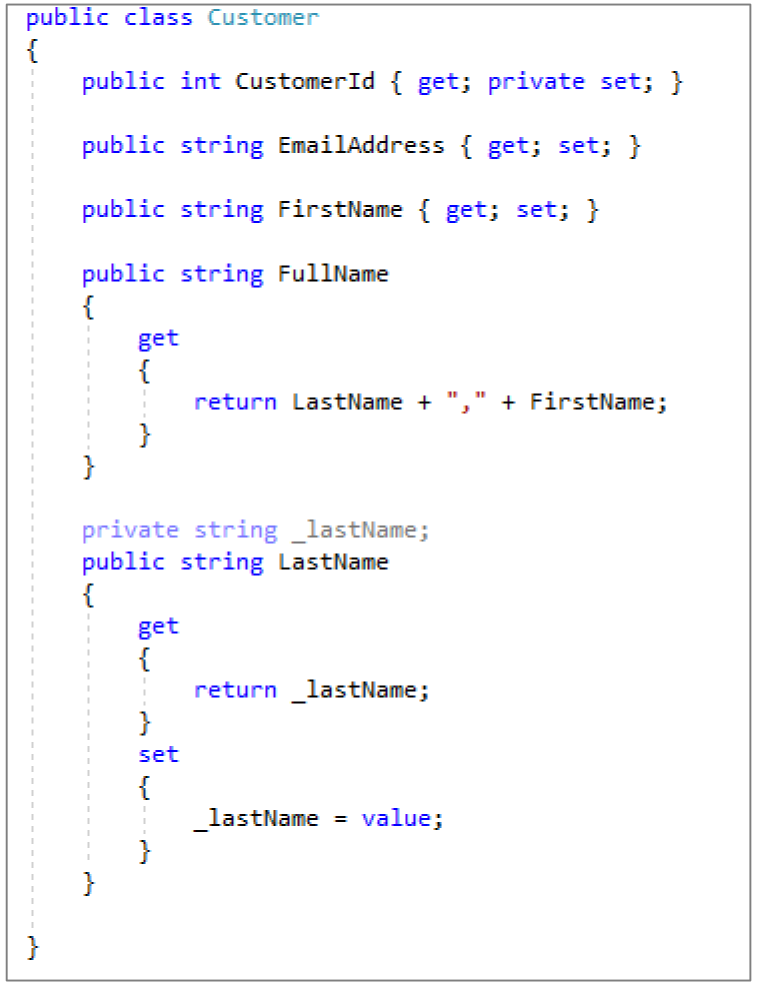
Customer

- Name
- Email address
- Home address
- Work address
- Validate()
- Retrieve()
- Save()

Demo

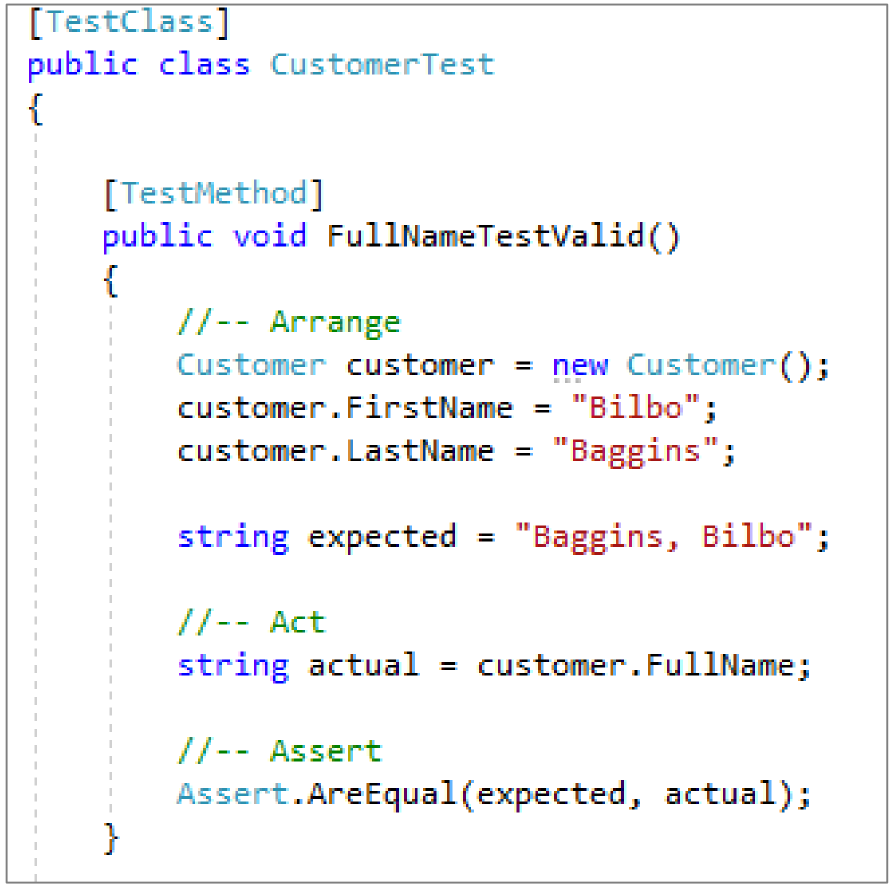
Using snippets

Testing Our Code



```
public class Customer
{
    public int CustomerId { get; private set; }

    public string EmailAddress { get; set; }

    public string FirstName { get; set; }

    public string FullName
    {
        get
        {
            return LastName + "," + FirstName;
        }
    }

    private string _lastName;
    public string LastName
    {
        get
        {
            return _lastName;
        }
        set
        {
            _lastName = value;
        }
    }

}
```

Our class



```
[TestClass]
public class CustomerTest
{

    [TestMethod]
    public void FullNameTestValid()
    {
        //-- Arrange
        Customer customer = new Customer();
        customer.FirstName = "Bilbo";
        customer.LastName = "Baggins";

        string expected = "Baggins, Bilbo";

        //-- Act
        string actual = customer.FullName;

        //-- Assert
        Assert.AreEqual(expected, actual);
    }
```

Test for our class

Demo

Testing our class: valid values

Demo

Testing our class: invalid values

Creating a New Object

```
Customer customer = new Customer();
```

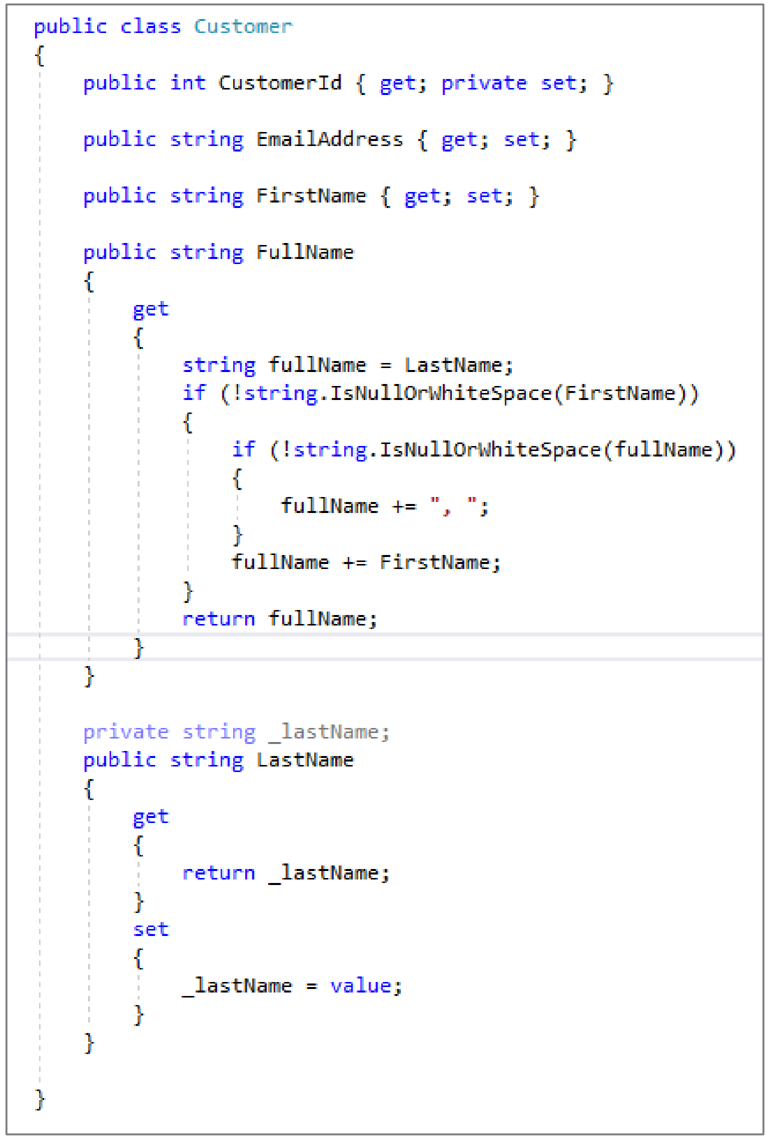
```
var customer = new Customer();
```

Accessing Properties

```
var customer = new Customer();
```

```
customer.LastName = "Baggins";
customer.FirstName = "Bilbo";
```

```
var actual = customer.FullName;
```



```
public class Customer
{
    public int CustomerId { get; private set; }

    public string EmailAddress { get; set; }

    public string FirstName { get; set; }

    public string FullName
    {
        get
        {
            string fullName = LastName;
            if (!string.IsNullOrWhiteSpace(FirstName))
            {
                if (!string.IsNullOrWhiteSpace(fullName))
                {
                    fullName += ", ";
                }
                fullName += FirstName;
            }
            return fullName;
        }
    }

    private string _lastName;
    public string LastName
    {
        get
        {
            return _lastName;
        }
        set
        {
            _lastName = value;
        }
    }

}
```

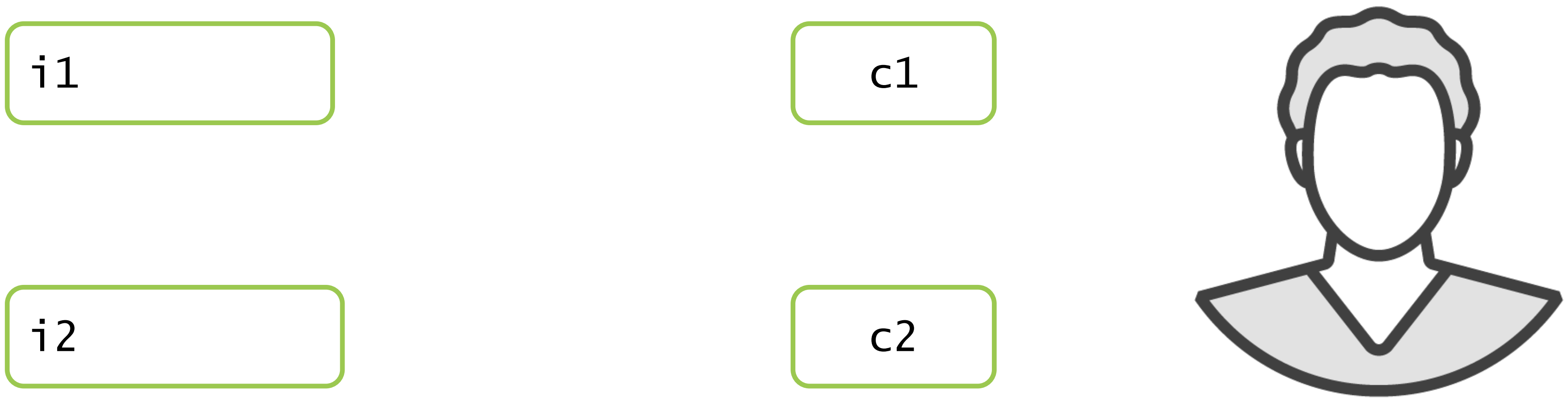
Objects Are Reference Types

```
int i1;
i1 = 42;

int i2 = i1;
i2 = 2;
```

What is i1?

```
var c1 = new Customer();
c1.FirstName = "Bilbo";

var c2 = c1;
c2.FirstName = "Frodo";
```

What is c1.FirstName?



Static Modifier

```
public static int InstanceCount { get; set; }
```

```
Customer.InstanceCount += 1;
```

Layering the Application

User interface

Business logic

Data access

Common library

Building a Class

Each class defines a type

Give the class a good name

Set the appropriate access modifier

```
public class Customer
{

}
```

Defining Properties: Manually

Declare the backing field

Declare the property

Add the getter and setter

```
private string _lastName;
public string LastName
{
    get
    {
        return _lastName;
    }
    set
    {
        _lastName = value;
    }
}
```

Defining Properties: Auto-implemented

Manages the backing field automatically

Use when the setter and getter don’t need logic

```
public string FirstName { get; set; }
```

Use Visual Studio snippets

Unit Testing

Create a separate project

Set a reference to the business layer component

Define tests for valid and invalid scenarios

Organize the test

- Arrange: Set up the test
- Act: Access the member being tested
- Assert: Determine the result

Working with Objects

```
Customer customer = new Customer();
```

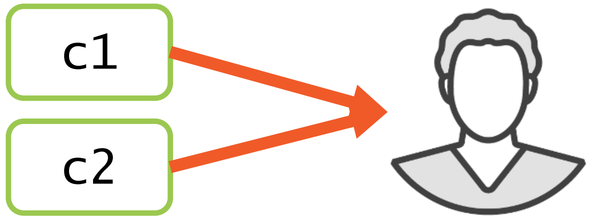
Creating an object

```
var customer = new Customer();
```

var keyword

```
customer.FirstName = "Bilbo";
```

Setting properties



Objects are reference types

```
public static int InstanceCount { get; set; }
Customer.InstanceCount += 1;
```

Static modifier

Customer Class

Customer

- Name
- Email address
- Home address
- Work address
- Validate()
- Retrieve()
- Save()