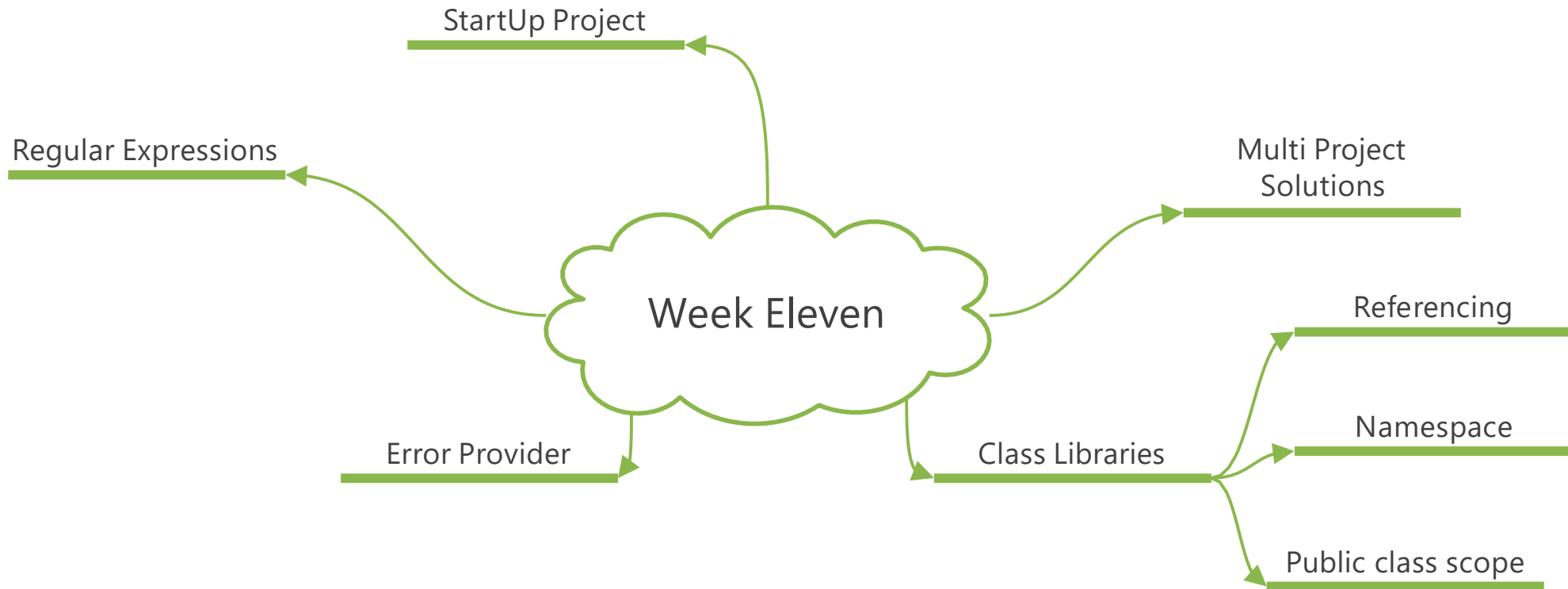


# COMP 3602

C# Application Development  
Week Eleven



# Tonight's Learning Outcomes



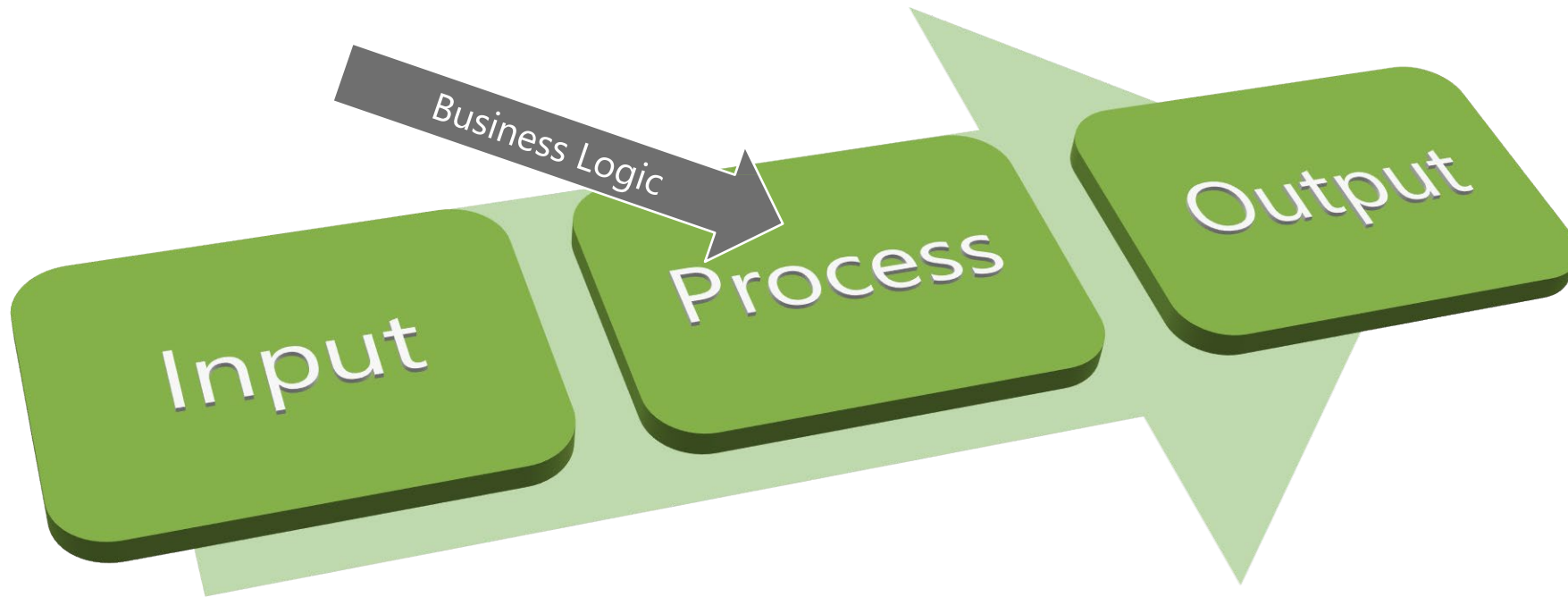
# Survey

Please take a few minutes to fill out the course survey.

# Decoupling UI from Business Logic

## Data Processing Stages

```
int value1 = 50;    int total;  
int value2 = 75;    total = value1 + value2;    labelResult.Text =  
                    total.ToString("N0");
```



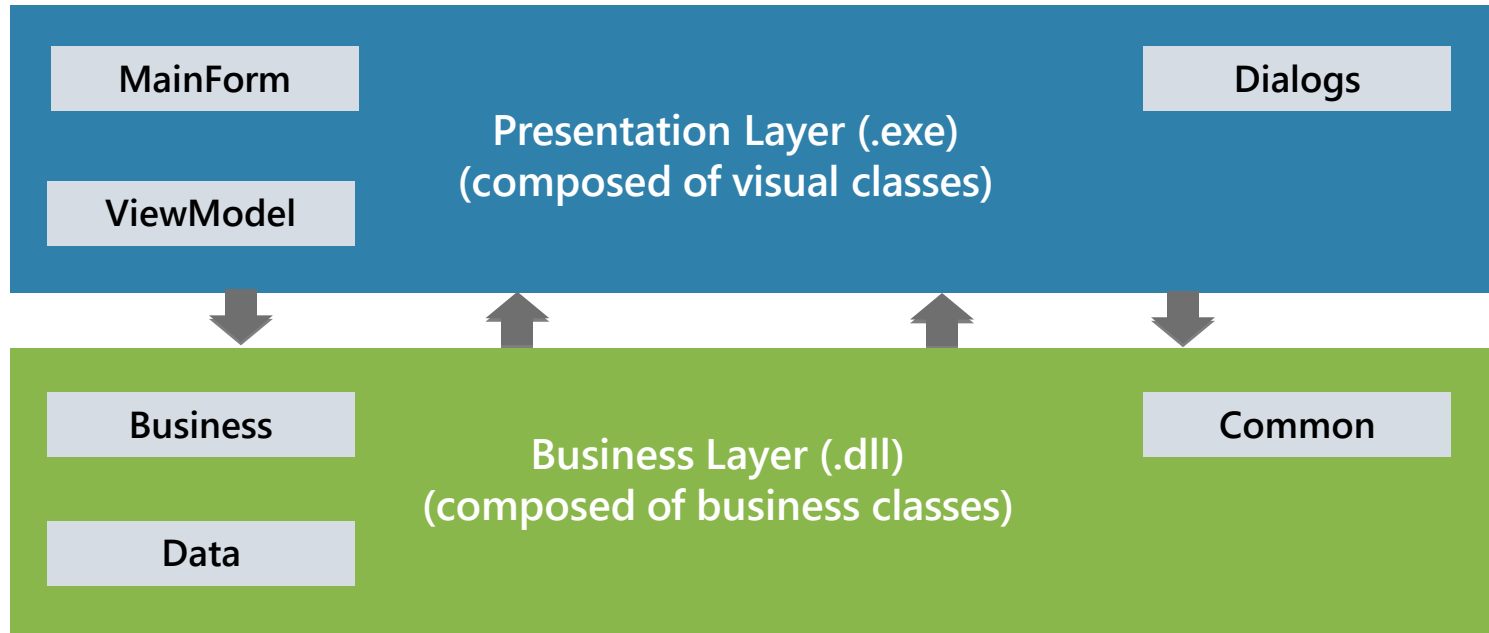
All software systems contain these three basic steps:

- 1) Get some input data
- 2) Do some operation on that data
- 3) Output the result

This is true whether the system is a Calculator, ERP System, Game, Mars Rover Guidance System or any other type of system.

# Decoupling UI from Business Logic

## Tiers and Layers



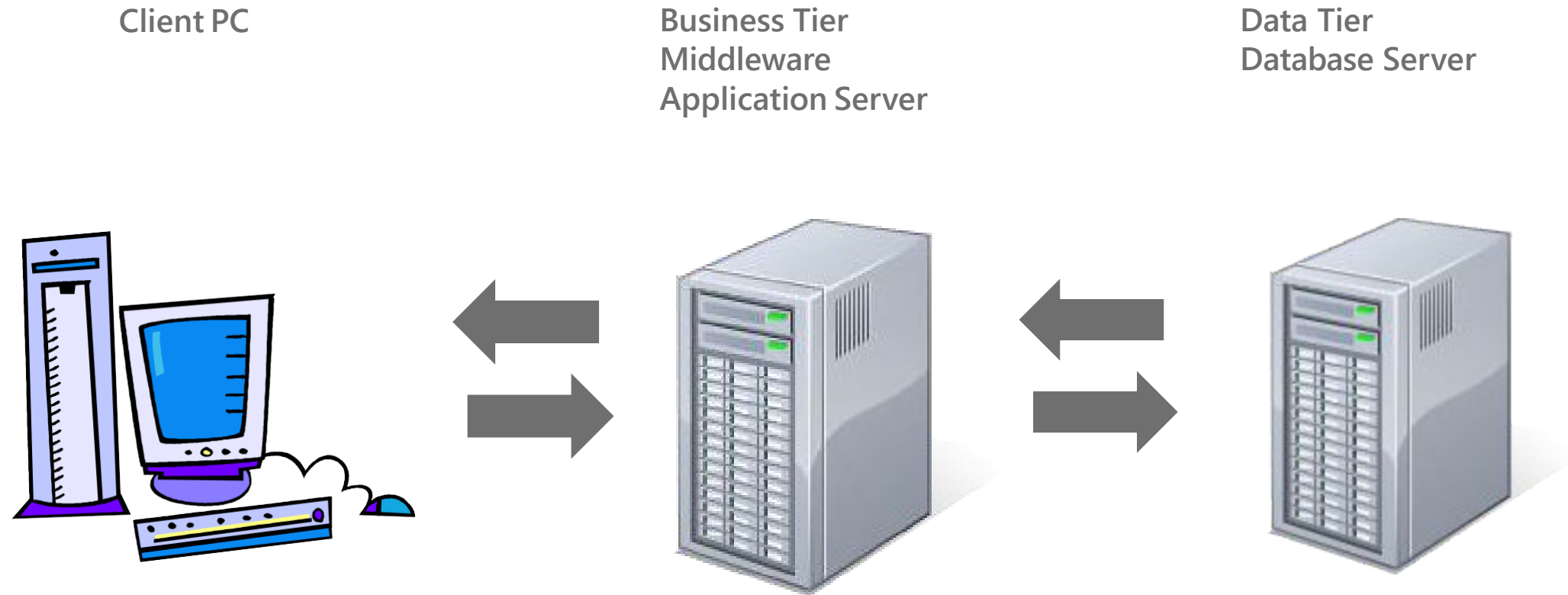
Decoupling the user interface from the business logic means moving the business logic *out* of the UI classes and *into* separate classes, known as business classes.

The business classes should not perform **ANY** user interface-related functions (not even a message box). This makes them more reusable.

The form, and other visual classes, make use of the business classes.

# Decoupling UI from Business Logic

## Tiers and Layers



# Help Desk System

## Original Design

### Windows Forms Application

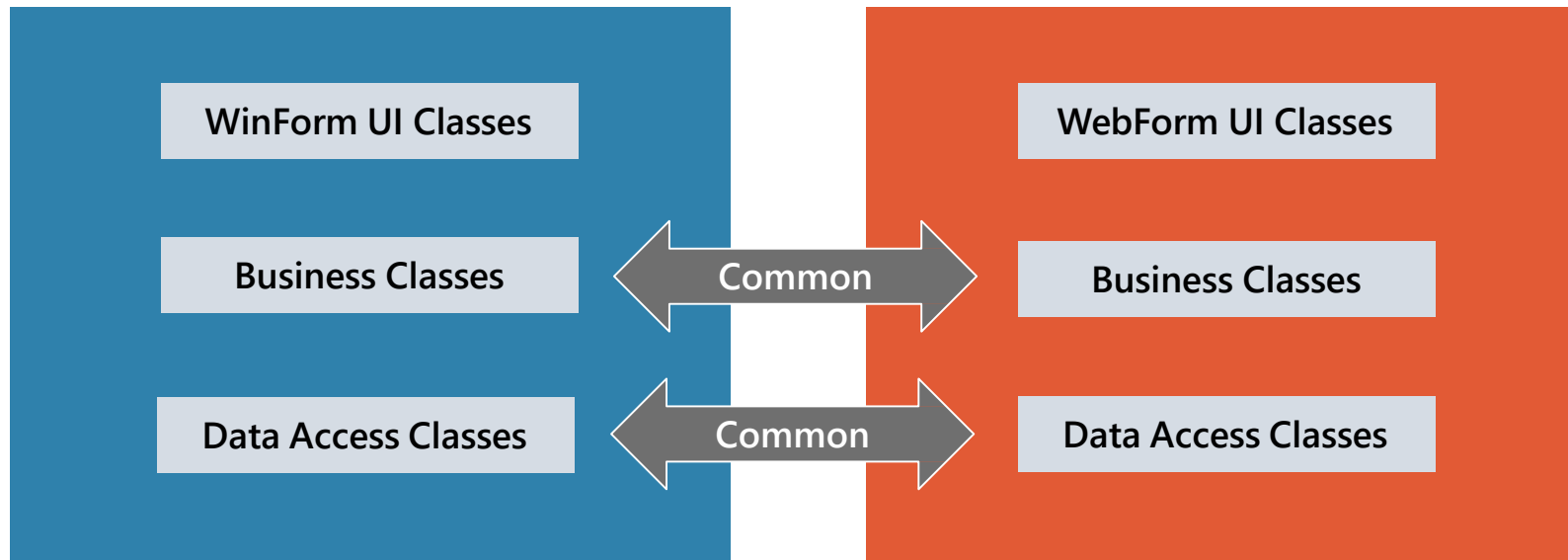


Original design places all classes in one large monolithic application.

This works, but has very poor extensibility as we will soon discover.

# Help Desk System

## Extended Functionality – Poor Maintainability



Building a Web version of the application would require a Web UI along with the same Business and Data Access code.

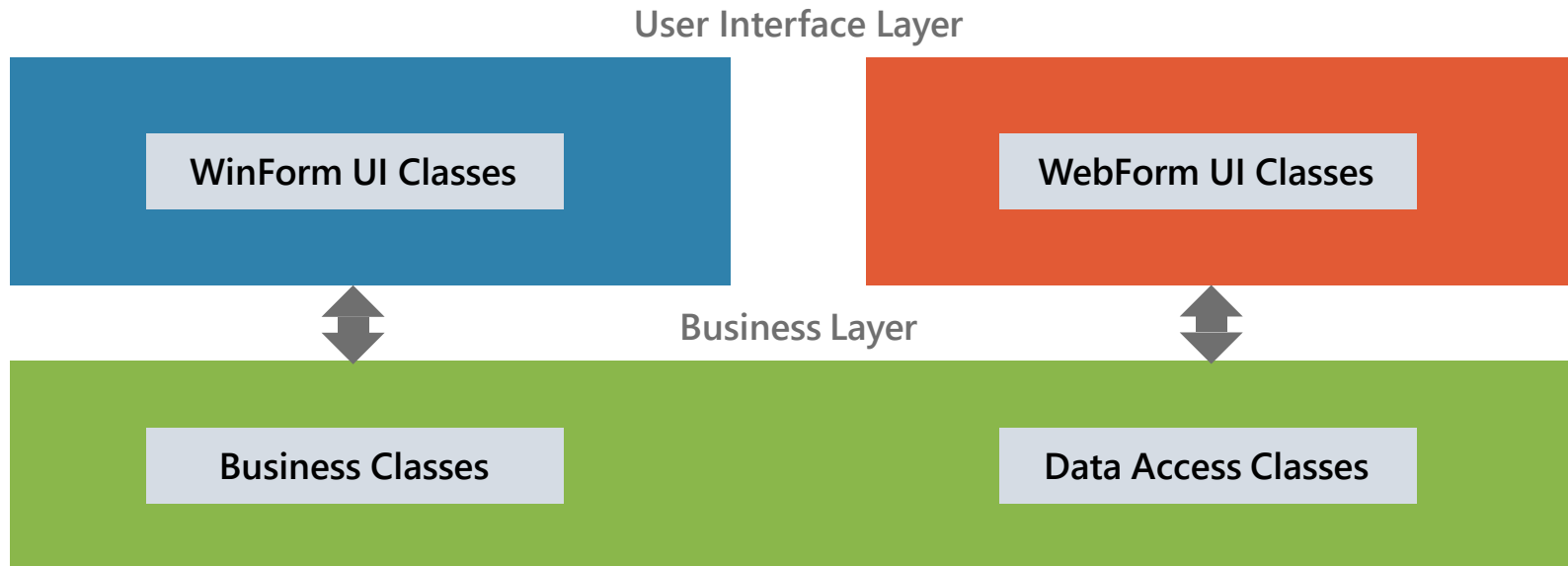
There would be a lot of copy/paste of the Business and Data Access code resulting in code duplication.

Maintenance would be challenging as it is hard to keep code changes in sync between the two versions.



# Help Desk System

## Extended Functionality – Improved Maintainability



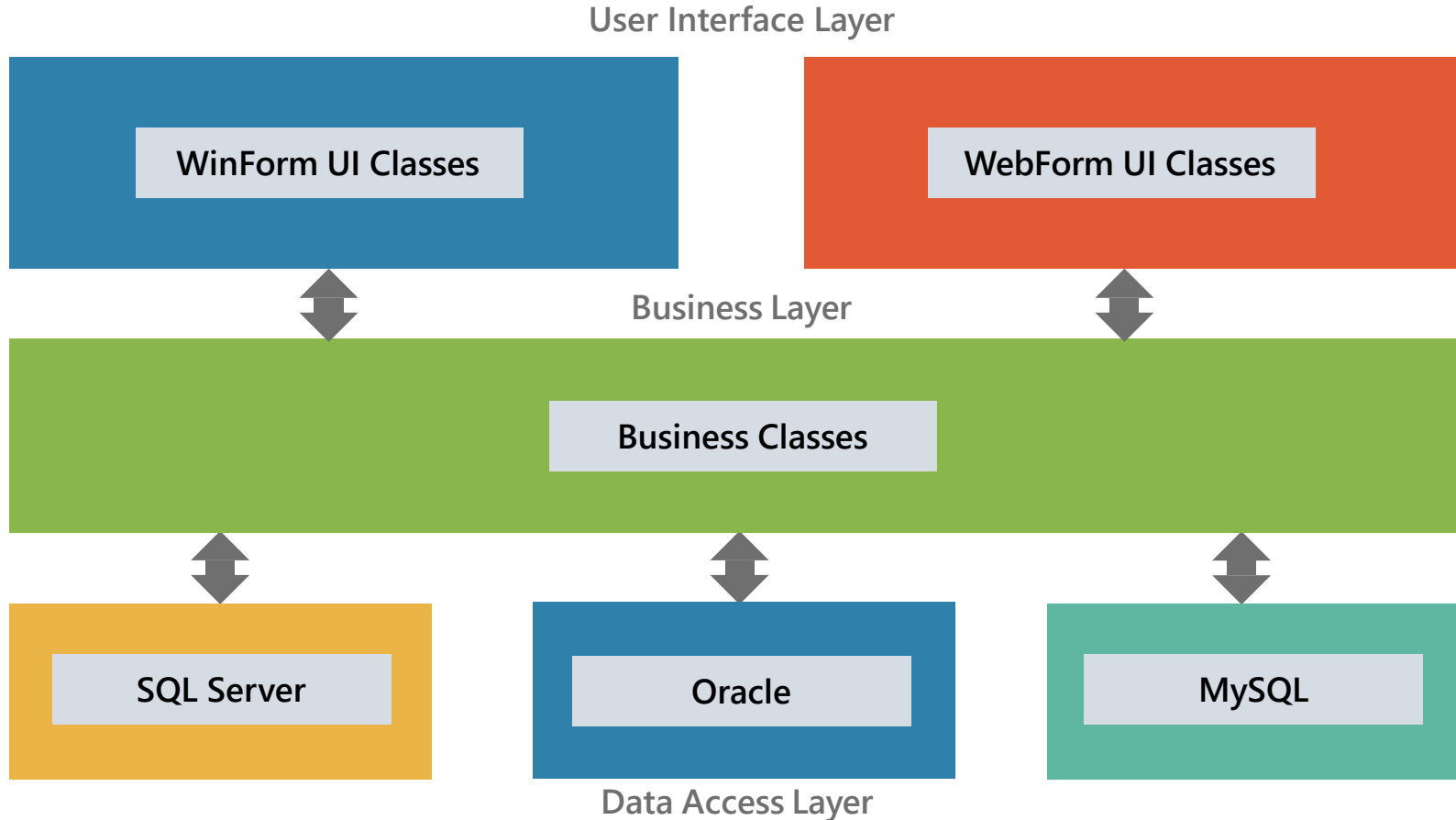
Moving the common code into a separate module will allow each UI to access the same code.

Code duplication is reduced and the business logic is consistent with each UI.

The module can be separated logically or physically, preferably.

# Help Desk System

## Extended Functionality – Enhanced Maintainability



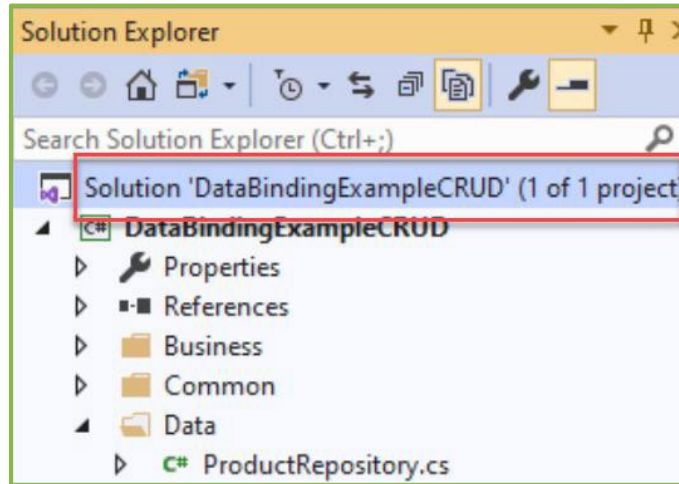
Separation of the Business classes from the Data Access can provide further flexibility.

We can now use a number of different data stores.

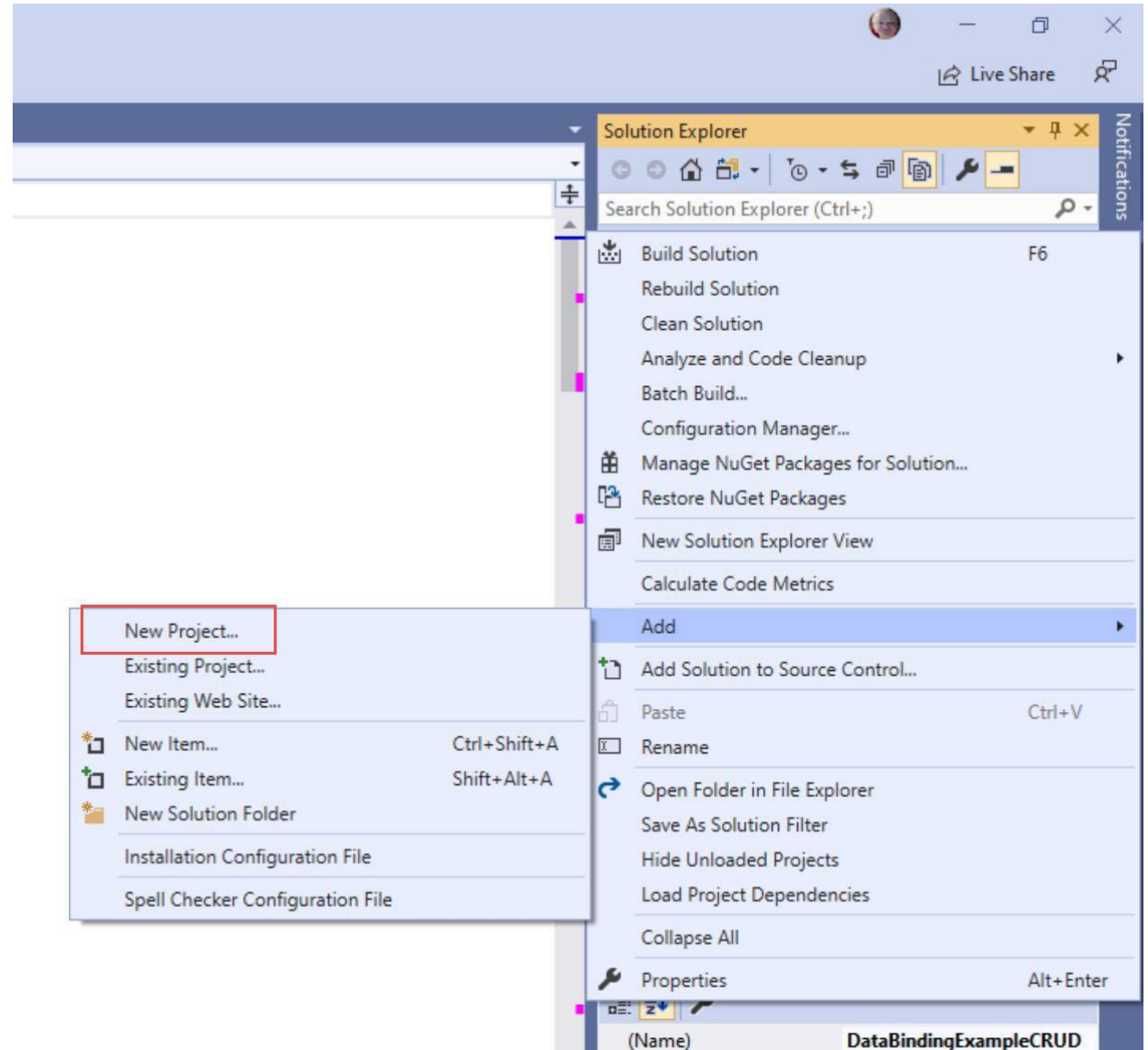
The Business layer is the “heart & sole” of your application. This is where the processes and business rules are defined and enforced.

# Class Libraries

## Creating

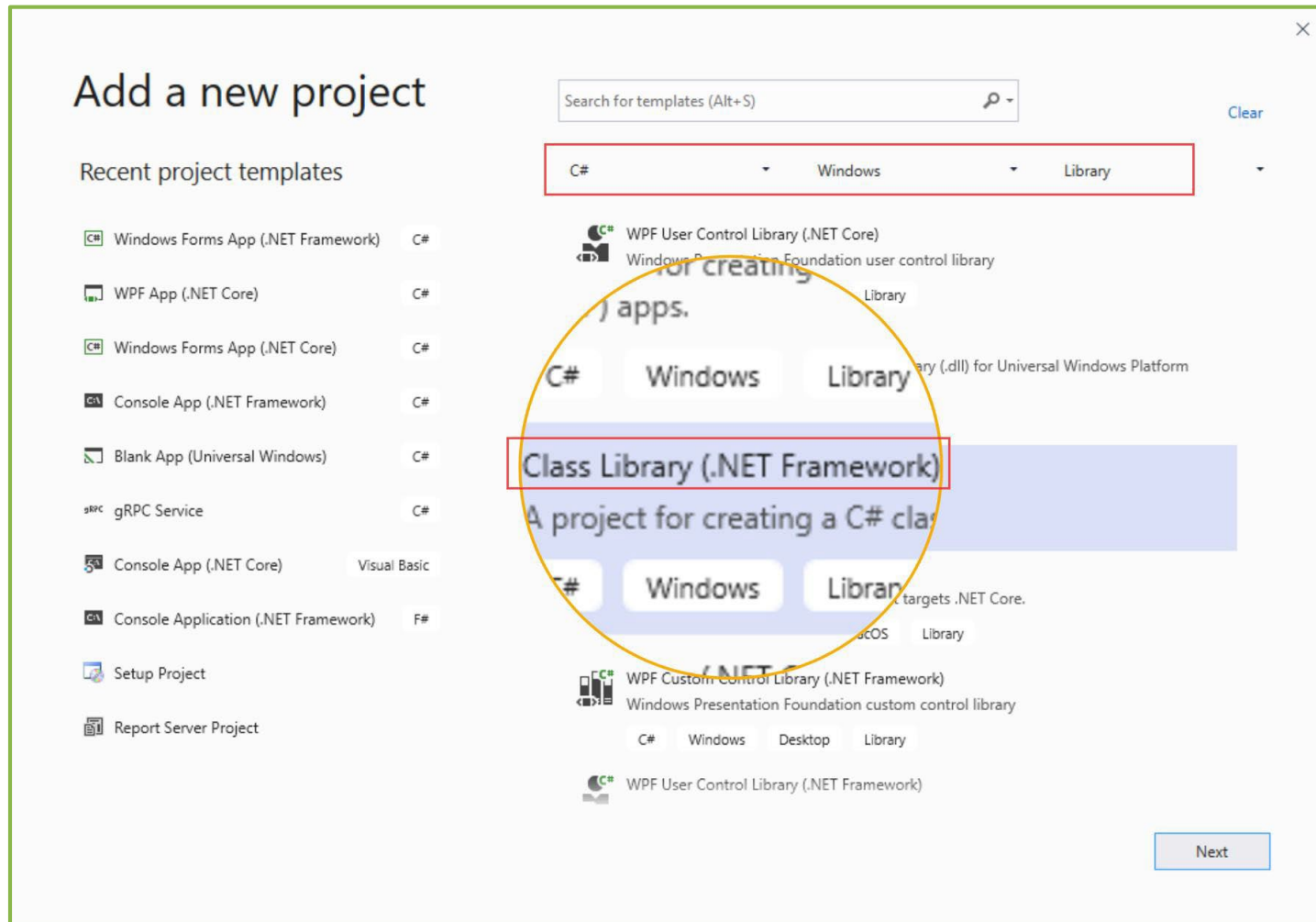


- Right-click on the solution in the Solution Explorer
- Click Add
- Click New Project



# Class Libraries

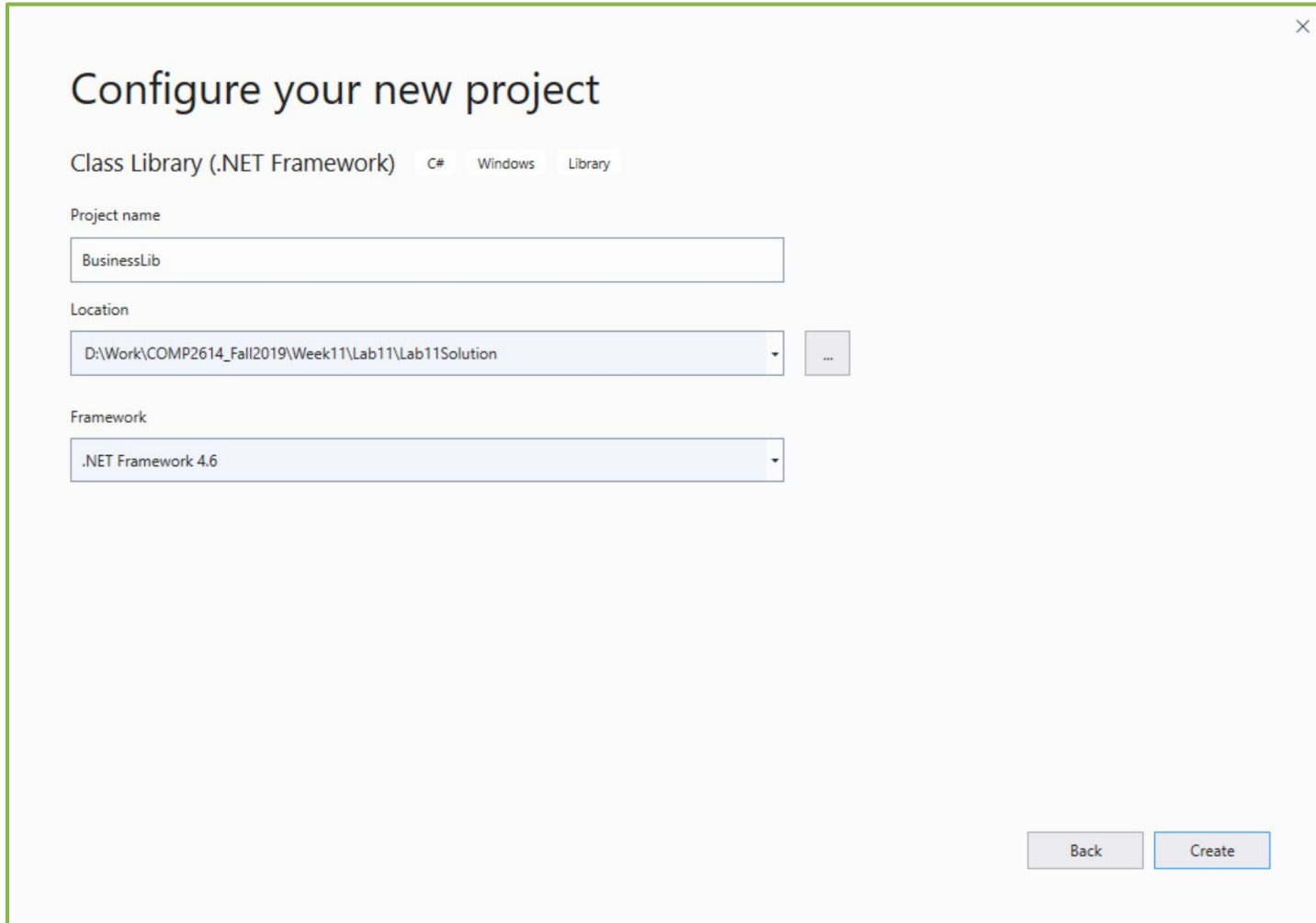
## Creating



- Filter Project Templates with C#/Windows/Library
- Select Class Library (.NET Framework)
- Click Next

# Class Libraries

## Creating



Configure your new project

Class Library (.NET Framework) C# Windows Library

Project name

BusinessLib

Location

D:\Work\COMP2614\_Fall2019\Week11\Lab11\Lab11Solution

Framework

.NET Framework 4.6

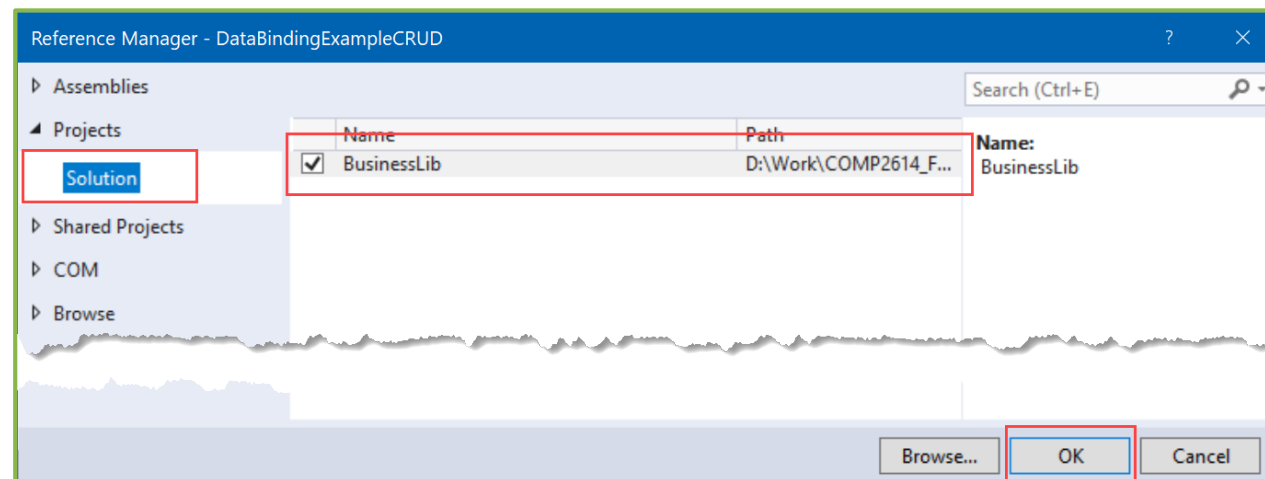
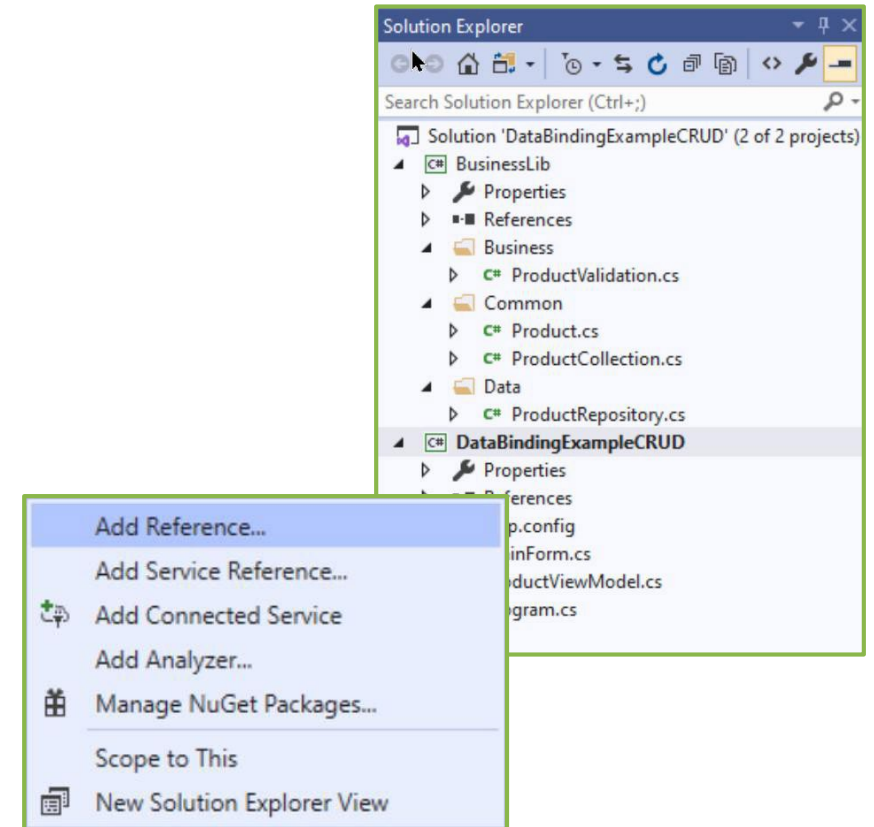
Back Create

- Name your project (PascalCase) This will become the DLL filename and default namespace for the project
- Leave the default location
- Select a Framework version. This version must be the same, or earlier than the .exe project

# Class Libraries

## Referencing

- Right-click on References in the UI Project in the Solution Explorer
- Click Add Reference...
- Select Solution under the Projects tab.
- Select the Class Libraries to add to the UI Project (In this case, just the one)
- Click OK



# Class Libraries

## Namespaces and Scoping

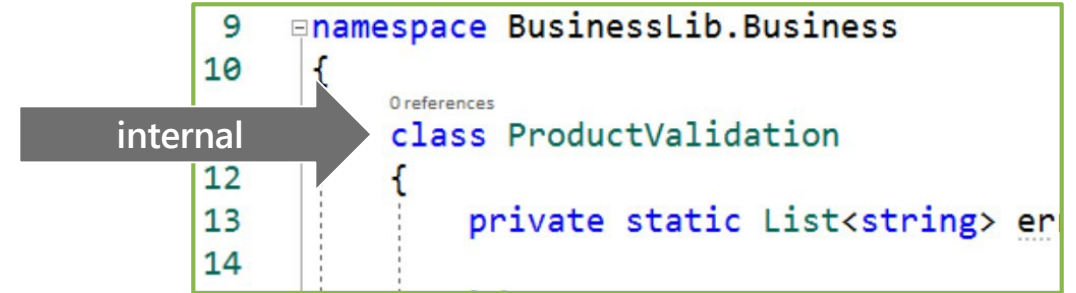
Add a using statement at the top of the each .exe Project source file that requires access to classes in the .dll Project

.exe Assembly

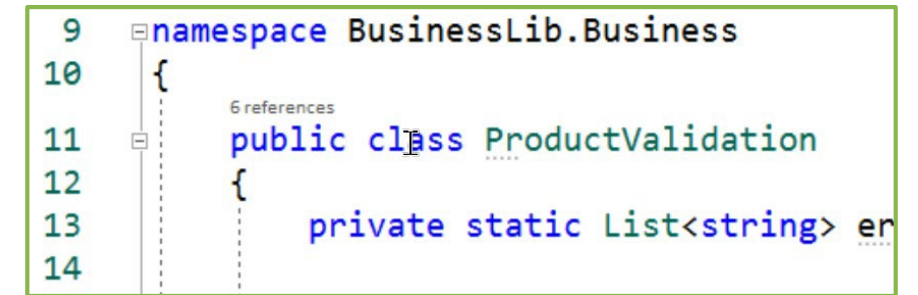
```
using dllnamespace;  
  
internal – visible within  
Assembly only  
  
AccountB is visible from  
.exe Project  
  
AccountA is not
```

.dll Assembly

```
class AccountA  
{  
    ...  
}  
  
public class AccountB  
{  
    ...  
}
```



```
9 namespace BusinessLib.Business  
10 {  
11     class ProductValidation  
12     {  
13         private static List<string> er  
14     }
```



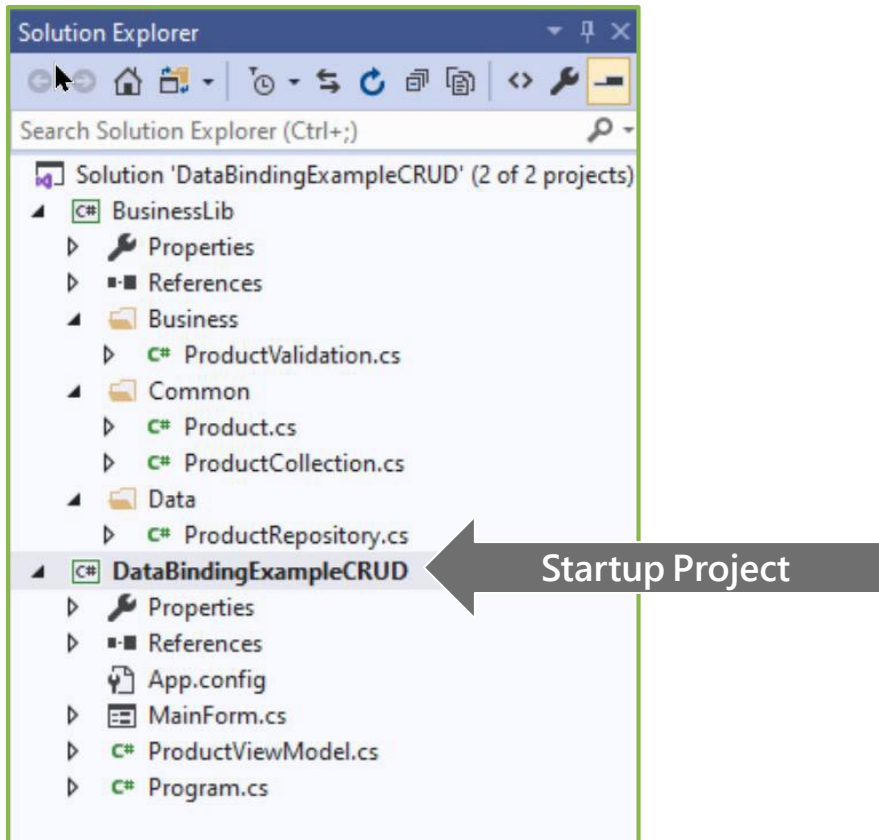
```
9 namespace BusinessLib.Business  
10 {  
11     public class ProductValidation  
12     {  
13         private static List<string> er  
14     }
```

Default scope for a class is internal, meaning it is visible only within its own assembly

A class must be explicitly defined as public to be visible outside of its own assembly

# Class Libraries

## Startup Project



- When your solution contains more than one project, only one of them is designated as the startup project, which is the project that runs when you start the debugger.
- Make sure that the EXE project is the startup project by right clicking on it and selecting "Set as StartUp Project". You will know that it is the startup project when it turns bold.
- The first project created in the solution is the default startup project.

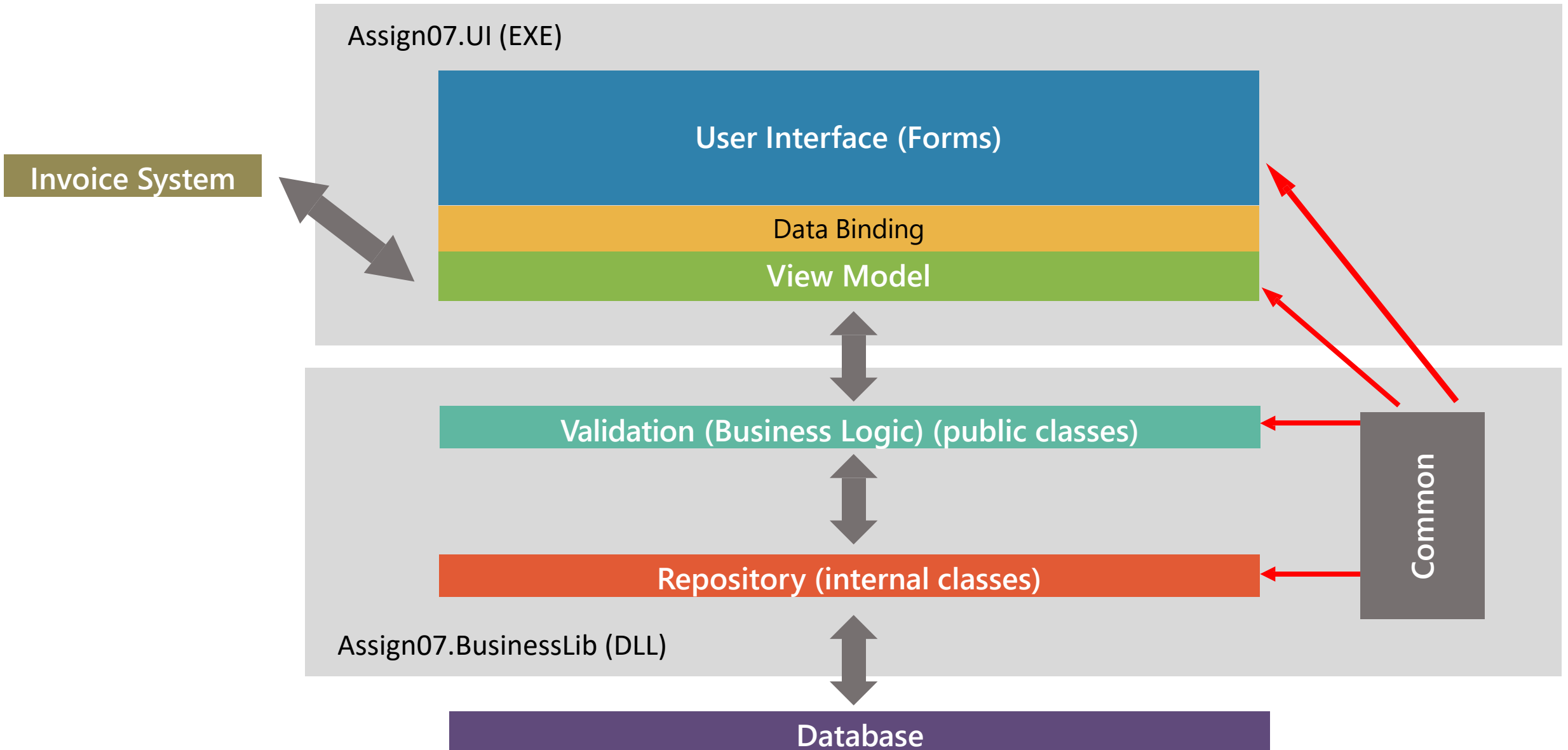


# Class Libraries

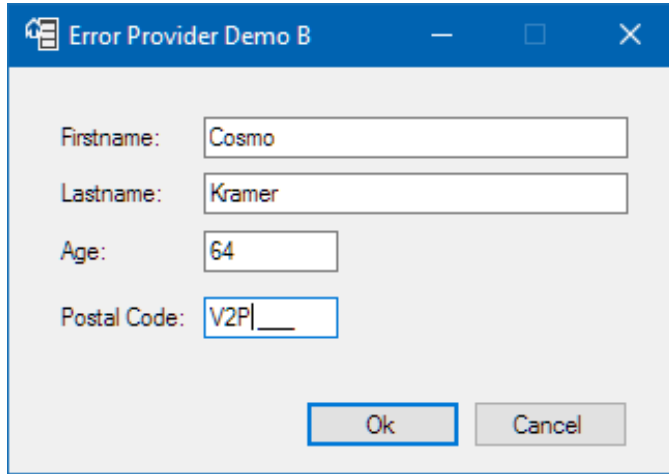
## Checklist

- Ensure the EXE project references the DLL project (check EXE reference list) ✓
- Ensure the EXE project references the correct DLL project ✓
- Ensure the EXE project file(s) have the appropriate 'using' statements to scope the DLL project namespace(s) ✓
- Ensure all DLL project classes that the EXE project accesses are explicitly marked public ✓

## Assignment 7 Architecture (Part D)



# MaskedTextBox Class



A screenshot of a Windows application window titled "Error Provider Demo B". The window contains four text input fields with labels to their left: "Firstname:" with the value "Cosmo", "Lastname:" with the value "Kramer", "Age:" with the value "64", and "Postal Code:" with the value "V2P|\_\_". The "Postal Code" field is highlighted with a blue border. At the bottom right of the window are "Ok" and "Cancel" buttons.

Specify a pattern in the Mask property of a MaskedTextBox and it will only accept specific characters at specific positions.

(like a Postal Code)

It cannot force the completion of the entry

Locked	False
Margin	3, 3, 3, 3
Mask	>L0L 0L0

Mask Property

# MaskedTextBox Class

Masking element	Description	Regex Mask Codes	Regular expression element
0	Any single digit between 0 and 9. Entry required.		\d
9	Digit or space. Entry optional.		[ \d]?
#	Digit or space. Entry optional. If this position is left blank in the mask, it will be rendered as a space. Plus (+) and minus (-) signs are allowed.		[ \d+-]?
L	ASCII letter. Entry required.		[a-zA-Z]
?	ASCII letter. Entry optional.		[a-zA-Z]?
&	Character. Entry required.		[\p{Ll}\p{Lu}\p{Lt}\p{Lm}\p{Lo}]
C	Character. Entry optional.		[\p{Ll}\p{Lu}\p{Lt}\p{Lm}\p{Lo}]?
A	Alphanumeric. Entry optional.		\W
.	Culture-appropriate decimal placeholder.		Not available.
,	Culture-appropriate thousands placeholder.		Not available.
:	Culture-appropriate time separator.		Not available.
/	Culture-appropriate date separator.		Not available.
\$	Culture-appropriate currency symbol.		Not available.
<	Converts all characters that follow to lowercase.		Not available.
>	Converts all characters that follow to uppercase.		Not available.
	Undoes a previous shift up or shift down.		Not available.
\	Escapes a mask character, turning it into a literal. "\\" is the escape sequence for a backslash.		\
All other characters.	Literals. All non-mask elements will appear as themselves within <a href="#">MaskedTextBox</a> .		All other characters.

# Regular Expressions

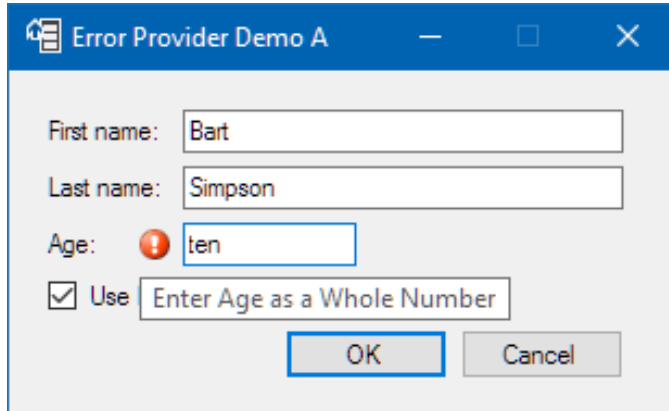
```
7 using System.Text;
8 using System.Windows.Forms;
9
10 using System.Text.RegularExpressions;
11
```

Add Namespace

Regular Expressions  
can be used to validate  
simple and complex  
patterns.

```
52 string regExCdnPostalCode = @"^[A-Z]\d[A-Z] \d[A-Z]\d$";
53
54 if (!Regex.IsMatch(maskedTextBoxPostalCode.Text, regExCdnPostalCode))
55 {
56     errorProvider.SetError(maskedTextBoxPostalCode, "Postal Code Format is Incorrect");
57 }
58 else
59 {
60     errorProvider.SetError(maskedTextBoxPostalCode, string.Empty);
61 }
62
```

# ErrorProvider Class



HideSelection	True
IconAlignment on errorProvider	MiddleRight
IconPadding on errorProvider	3
ImeMode	NoControl

The ErrorProvider can indicate errors far less obtrusively than a MessageBox.

Call the static method SetError passing a Control Name and a message and a glyph will appear next to that Control with the message as a ToolTip.

One ErrorProvider can service an entire Form.

```
74
75     if (!int.TryParse(textBoxAge.Text, out int age))
76     {
77         errorProvider.SetError(textBoxAge, "Enter Age as a Whole Number");
78     }
79     else
80     {
81         errorProvider.SetError(textBoxAge, string.Empty);
82     }
83
```

# Final Exam Prep

- Separation of Concerns
  - What should a class be responsible for?
  - What should it know about?
- Design
  - Which classes do you need?
  - What should those classes do?
- Clean, readable code
  - Full, meaningful names
  - Intuitive flow
  - Comments where appropriate
- C# Fluency
  - Naming conventions
  - Types/Casting/Converting/Parsing
  - Collections (Generics)
  - Properties (automatic, calculated, readonly, etc)
  - LINQ/Expression-bodied methods/properties
- ADO.Net
  - Connecting to and working with SQL
- Windows Forms UI
  - Keyboard navigation (tab order/mnemonics)
  - Working with Controls
    - Setting properties (in code and in designer)
    - Handling Events
  - Databinding
  - View Model design
  - Validation