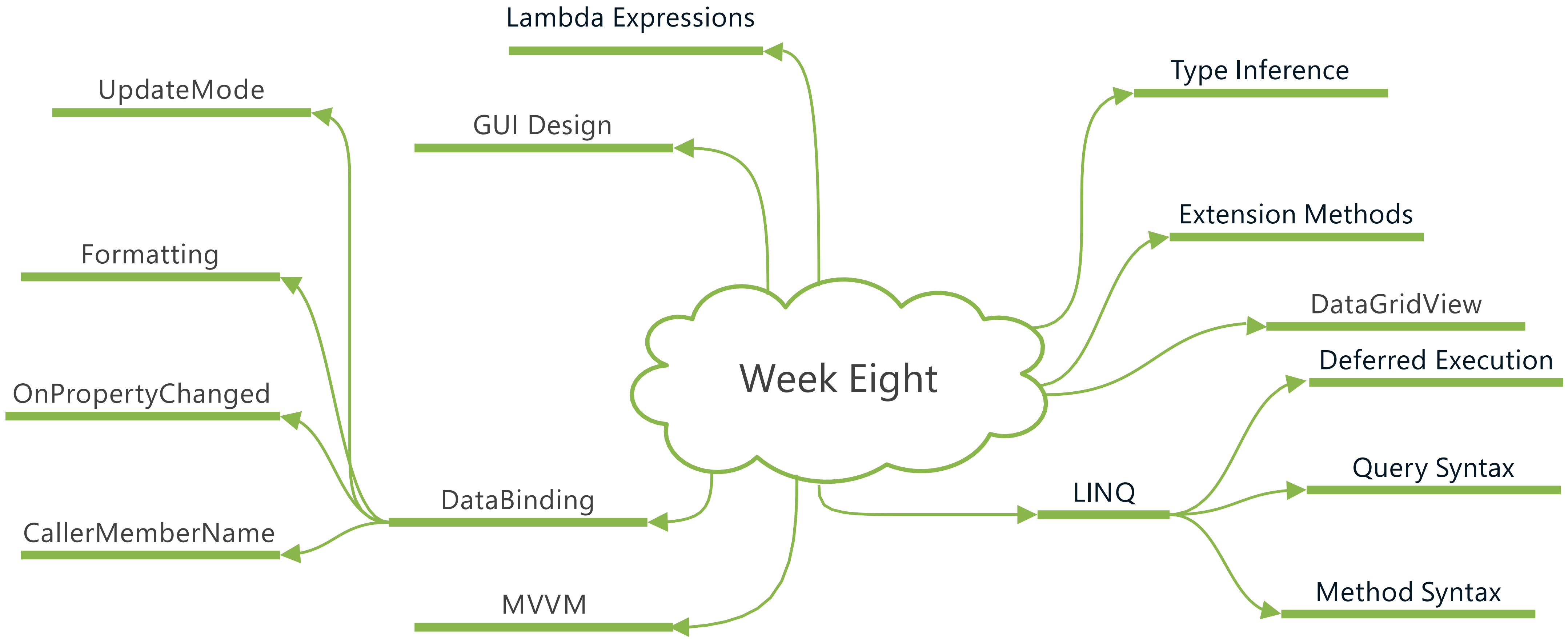
COMP 3602

C# Application Development

Week Eight

Tonight's Learning Outcomes



Assignment 05 Notes

- Public List<Customer> GetCustomers(string provinceCode = null)
  - There is so much overlapping code, that this is a great candidate to have the code for getting all customers and the code for the filtered list together in one method.
  - Using a default parameter means no one using the method needs to know what the “Magic Value” is that will result in all customers

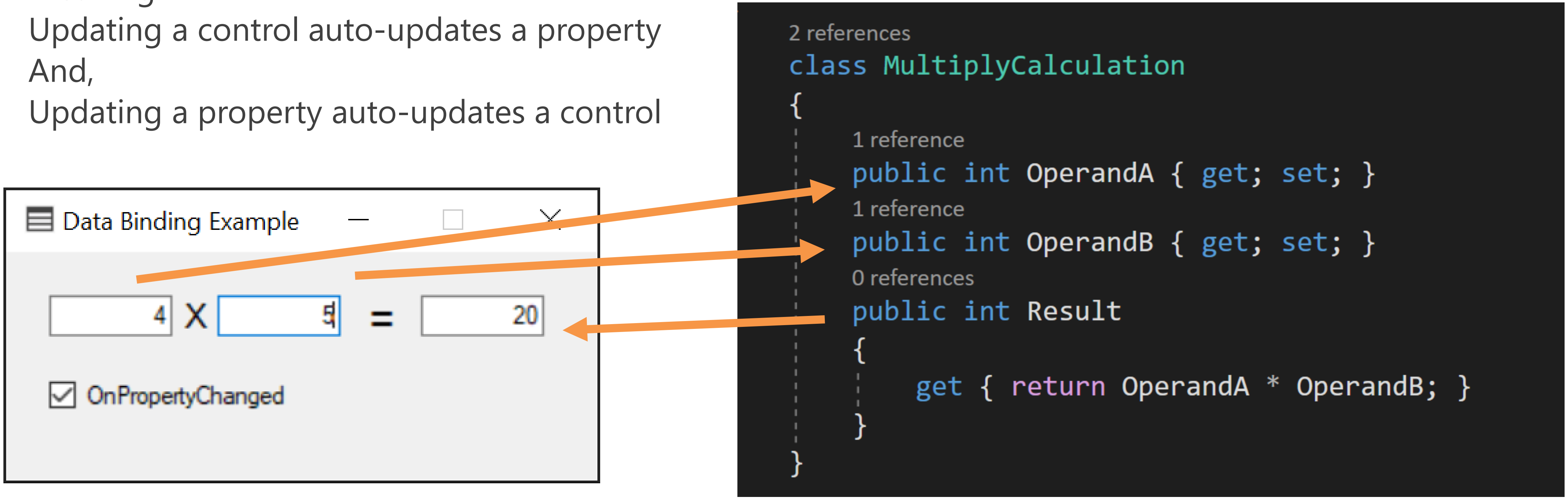
- CreditHold
  - What do we do with a nullable Boolean?
    - Make it a nullable property - bool? – Nullable<bool>
    - Keep it at the default value - False

- Exceptions in the Repository class.
  - What do we do with them?
    - Write it out to the Console
    - Try/catch – ignore and return nothing or a partial list
    - Do nothing in the Repository class and let it “bubble up” to the calling code.
- Where/how did you sort?
  - A) In the SQL statement?
  - B) Writing a CompareTo() method?
  - C) Both

DataBinding - Overview

Main goal: Have updates to controls and updates to a property update both the control and the property

Meaning:
Updating a control auto-updates a property
And,
Updating a property auto-updates a control



Data Binding Example

4 X 5 = 20

☑ OnPropertyChanged

```
2 references
class MultiplyCalculation
{
        1 reference
        public int OperandA { get; set; }
        1 reference
        public int OperandB { get; set; }
        0 references
        public int Result
        {
            get { return OperandA * OperandB; }
        }
}
```

Data Binding – Demo

Demo A (Multiply Calculator):

- Plain classes with properties work with databinding automatically
- The events are raised for us

Demo B (Total Calculator):

- With Collections, we need a way to know if the collection has changed
- We could just manually refresh the list each time, but this isn't ideal: easy to forget, inefficient

Demo C (Total Calculator):

- BindingList<T> will raise events on changes for us
- But, we may not have access to the data classes, and it isn't the best practice to modify them for application-specific needs like this
- Also, our TotalAmount still won't update

Demo D (Total Calculator):

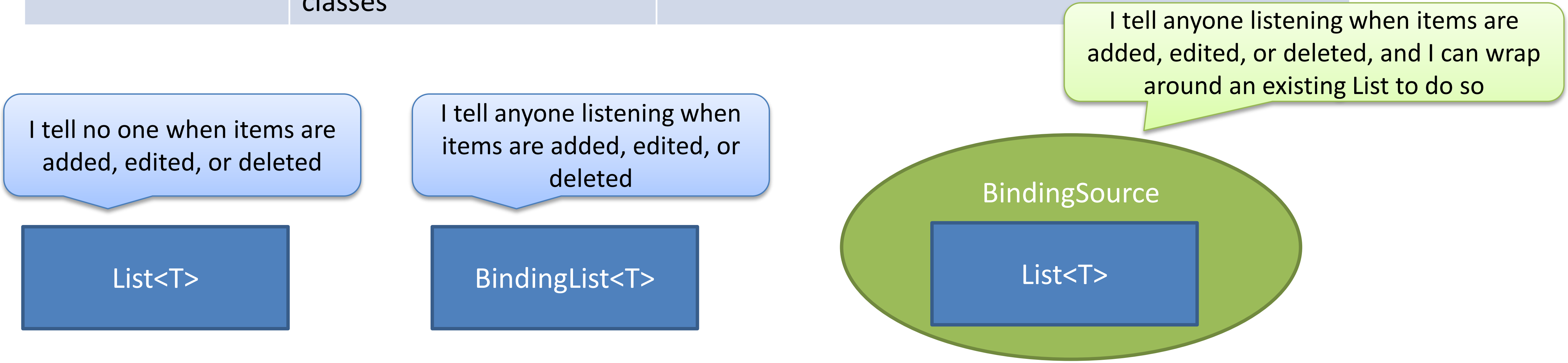
- We can get TotalAmount working by implementing INotifyPropertyChanged

Demo E (Total Calculator):

- Using a BindingSource to wrap around our List<T> is the best of both worlds
- This was more work to set up, but simpler to work with – less things to remember to do

Databinding with Collections

Method	Pros	Cons
List<T>	Easy	No events raised with changes – add, edit, delete
BindingList<T>	Easy, Raises events	We may not have access to data classes to change the type of List, not great SoC
BindingSource	Easy, Raises events, no need to change data classes	Object not typed – opens the door to adding invalid objects



I tell no one when items are added, edited, or deleted

List<T>

I tell anyone listening when items are added, edited, or deleted

BindingList<T>

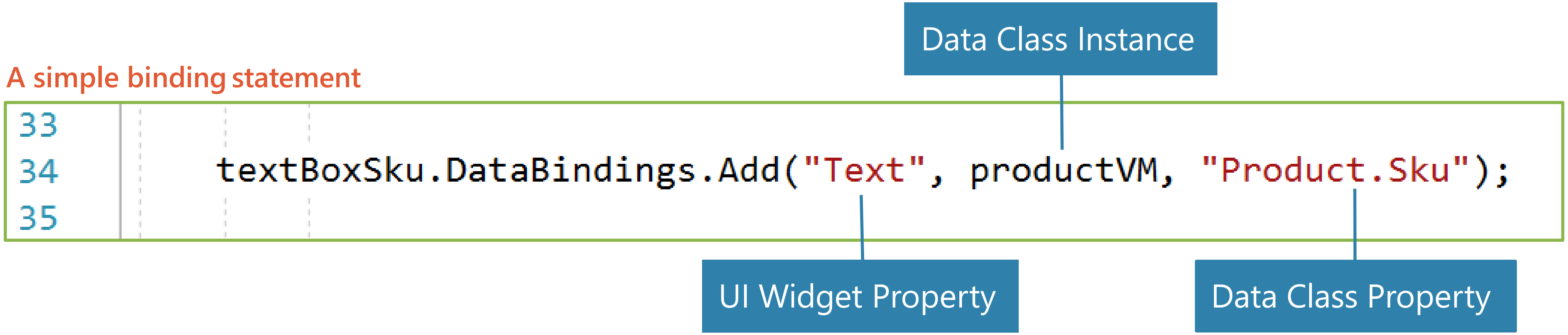
I tell anyone listening when items are added, edited, or deleted, and I can wrap around an existing List to do so

BindingSource

List<T>

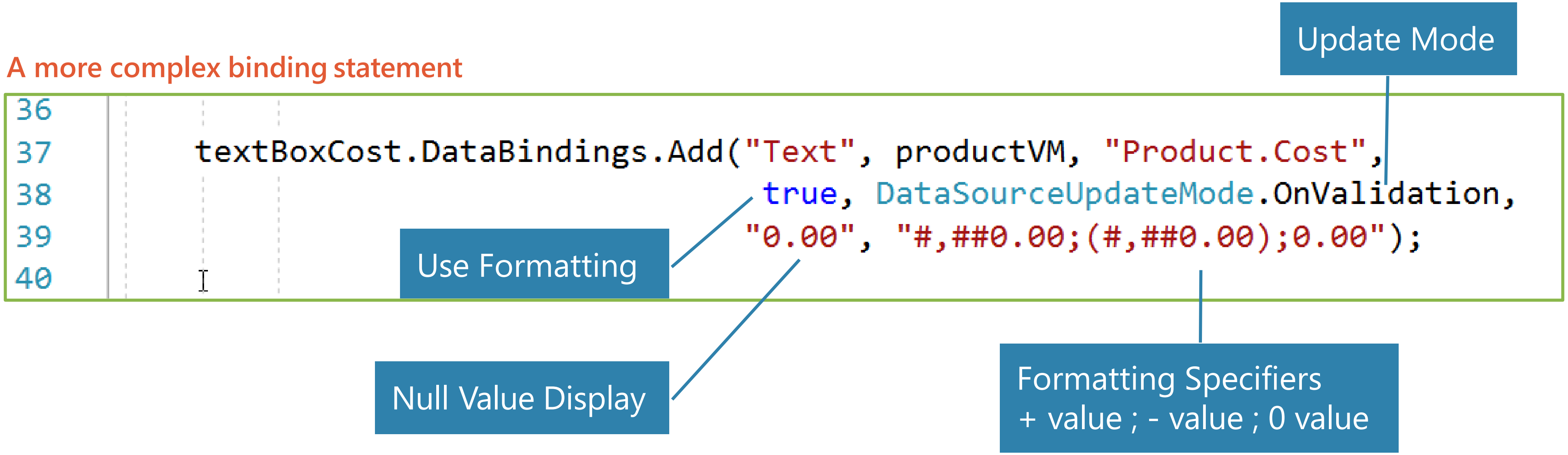
Data Binding – Binding to GUI Controls

A simple binding statement



```
        textBoxSku.DataBindings.Add("Text", productVM, "Product.Sku");
```

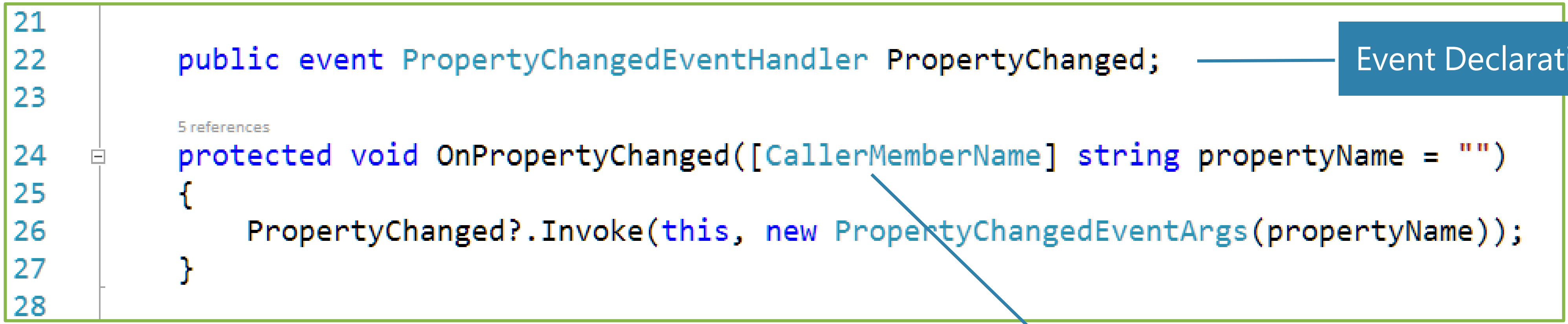
A more complex binding statement



```
        textBoxCost.DataBindings.Add("Text", productVM, "Product.Cost",
                                     true, DataSourceUpdateMode.OnValidation,
                                    "0.00", "#,##0.00;(#,##0.00);0.00");
```

Data Binding - OnPropertyChanged

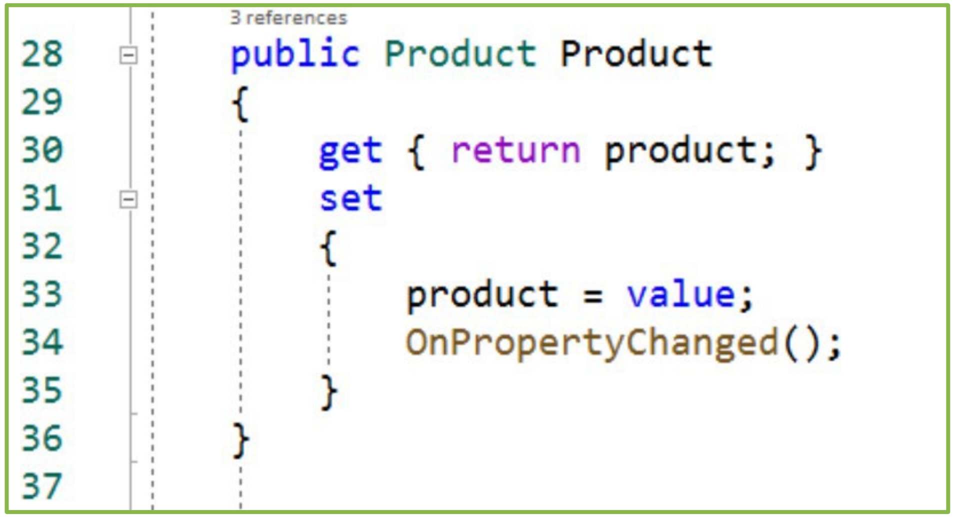
Event declaration and wrapper method to fire it



```
public event PropertyChangedEventHandler PropertyChanged;

5 references
protected void OnPropertyChanged([CallerMemberName] string propertyName = "")
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
```

Event Declaration

Method called in set block of Property



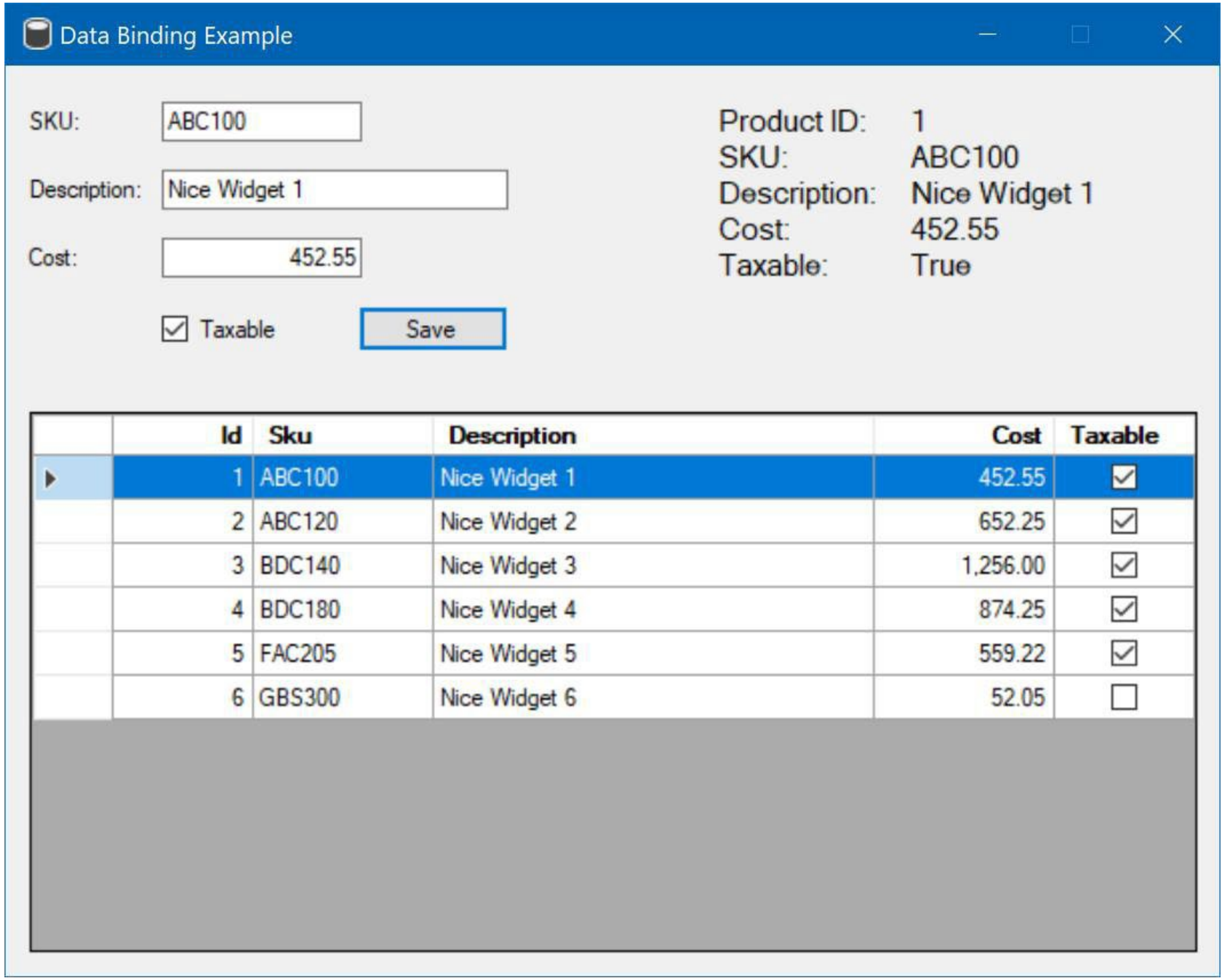
```
3 references
public Product Product
{
    get { return product; }
    set
    {
        product = value;
        OnPropertyChanged();
    }
}
```

[CallerMemberName]

This attribute automatically passes the Property name to the OnPropertychanged method

(using System.Runtime.CompilerServices;)

DataBindingDemoF - DataGridView Class



SKU: ABC100

Description: Nice Widget 1

Cost: 452.55

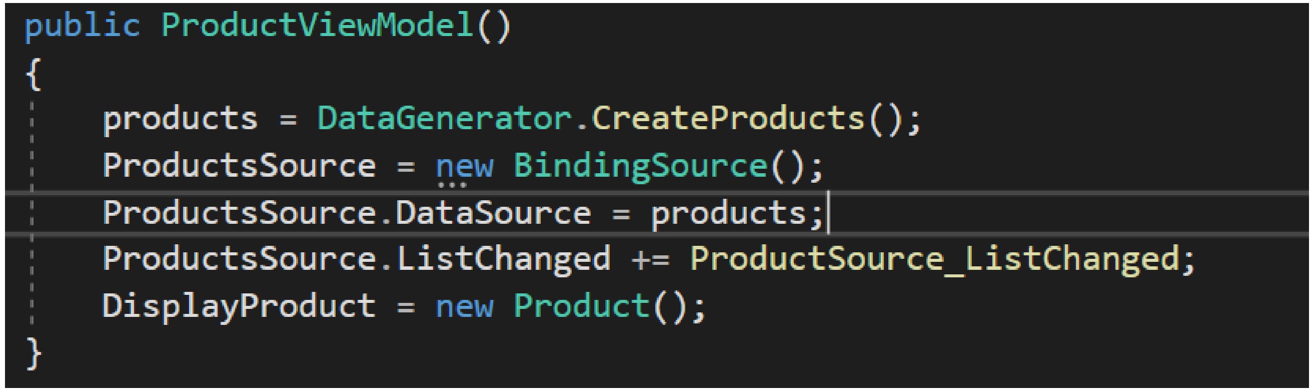
☑ Taxable

Product ID: 1
SKU: ABC100
Description: Nice Widget 1
Cost: 452.55
Taxable: True

Id	Sku	Description	Cost	Taxable
1	ABC100	Nice Widget 1	452.55	☑
2	ABC120	Nice Widget 2	652.25	☑
3	BDC140	Nice Widget 3	1,256.00	☑
4	BDC180	Nice Widget 4	874.25	☑
5	FAC205	Nice Widget 5	559.22	☑
6	GBS300	Nice Widget 6	52.05	□

- Can databind to a collection.
- Displays multiple columns.
- Columns can be automatically generated.
- Fully configurable.

Product ViewModel explained



```
public ProductViewModel()
{
    products = DataGenerator.CreateProducts();
    ProductsSource = new BindingSource();
    ProductsSource.DataSource = products;
    ProductsSource.ListChanged += ProductSource_ListChanged;
    DisplayProduct = new Product();
}
```

Initially empty new Product

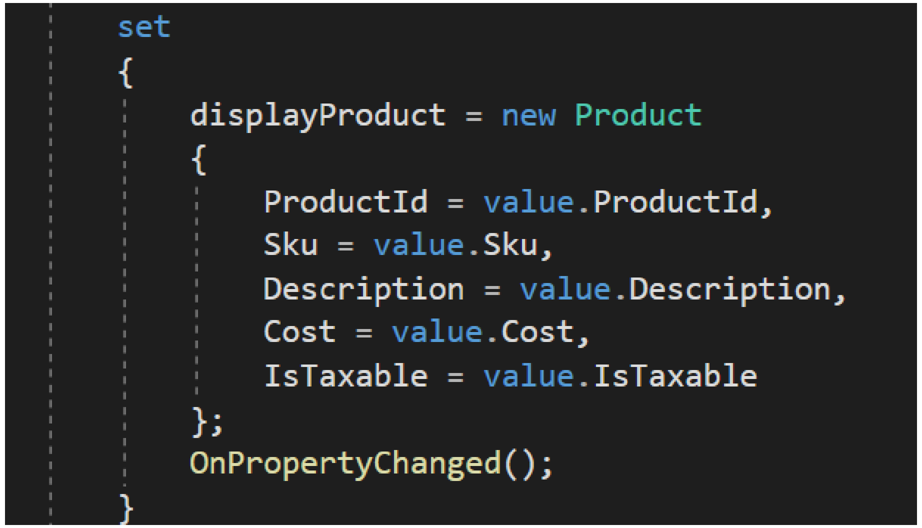
ProductViewModel.DisplayProduct

ProductViewModel.products

Array Index

0	1	10	ABC100	Nice Widget 1	452.55	true
1	2	5	ABC120	Nice Widget 2	652.25	true
2	Etc	….	…	..	..	..
3						
4						
5						

Product ViewModel explained



```
set
{
    displayProduct = new Product
    {
        ProductId = value.ProductId,
        Sku = value.Sku,
        Description = value.Description,
        Cost = value.Cost,
        IsTaxable = value.IsTaxable
    };
    OnPropertyChanged();
}
```

When selected, a **copy** of the product is created

ProductViewModel.DisplayProduct

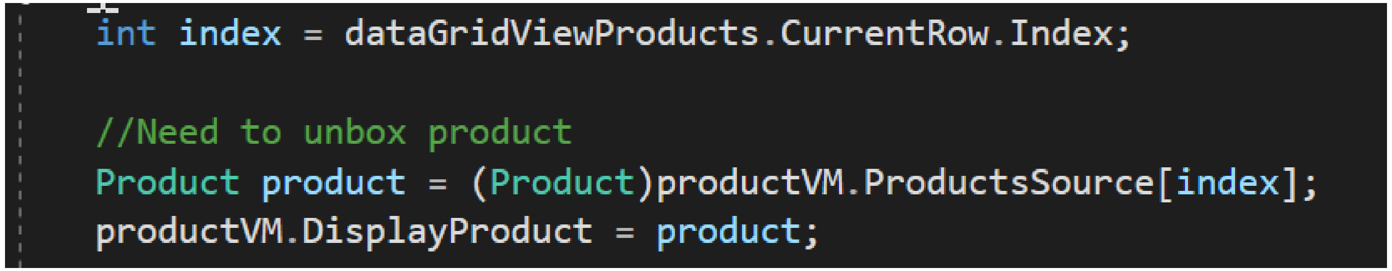
2	**5**	**ABC120**	**Nice Widget 1**	**652.25**	**true**

ProductViewModel.products

This item is selected

0	1	10	ABC100	Nice Widget 1	452.55	true
1	**2**	**5**	**ABC120**	**Nice Widget 2**	**652.25**	**true**
2	Etc	….	…	..	..	..
3						
4						
5						

Product ViewModel explained



```
int index = dataGridViewProducts.CurrentRow.Index;

//Need to unbox product
Product product = (Product)productVM.ProductsSource[index];
productVM.DisplayProduct = product;
```

When the copy is updated, the original object is **not** updated

ProductViewModel.DisplayProduct

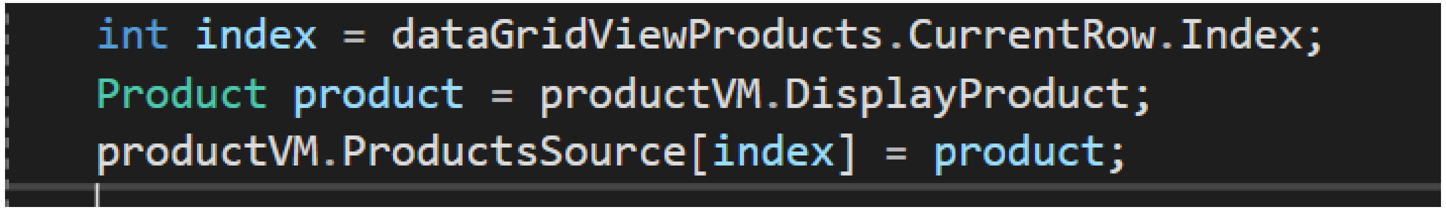
2	**500**	**ABC120**	**Nice Widget 1000**	**800.00**	**true**

ProductViewModel.products

0	1	10	ABC100	Nice Widget 1	452.55	true
1	**2**	**5**	**ABC120**	**Nice Widget 2**	**652.25**	**true**
2	Etc	….	…	..	..	..
3						
4						
5						

This item is not affected

Product ViewModel explained



```
int index = dataGridViewProducts.CurrentRow.Index;
Product product = productVM.DisplayProduct;
productVM.ProductsSource[index] = product;
```

So we need to remember to update the list and persist our change

ProductViewModel.DisplayProduct

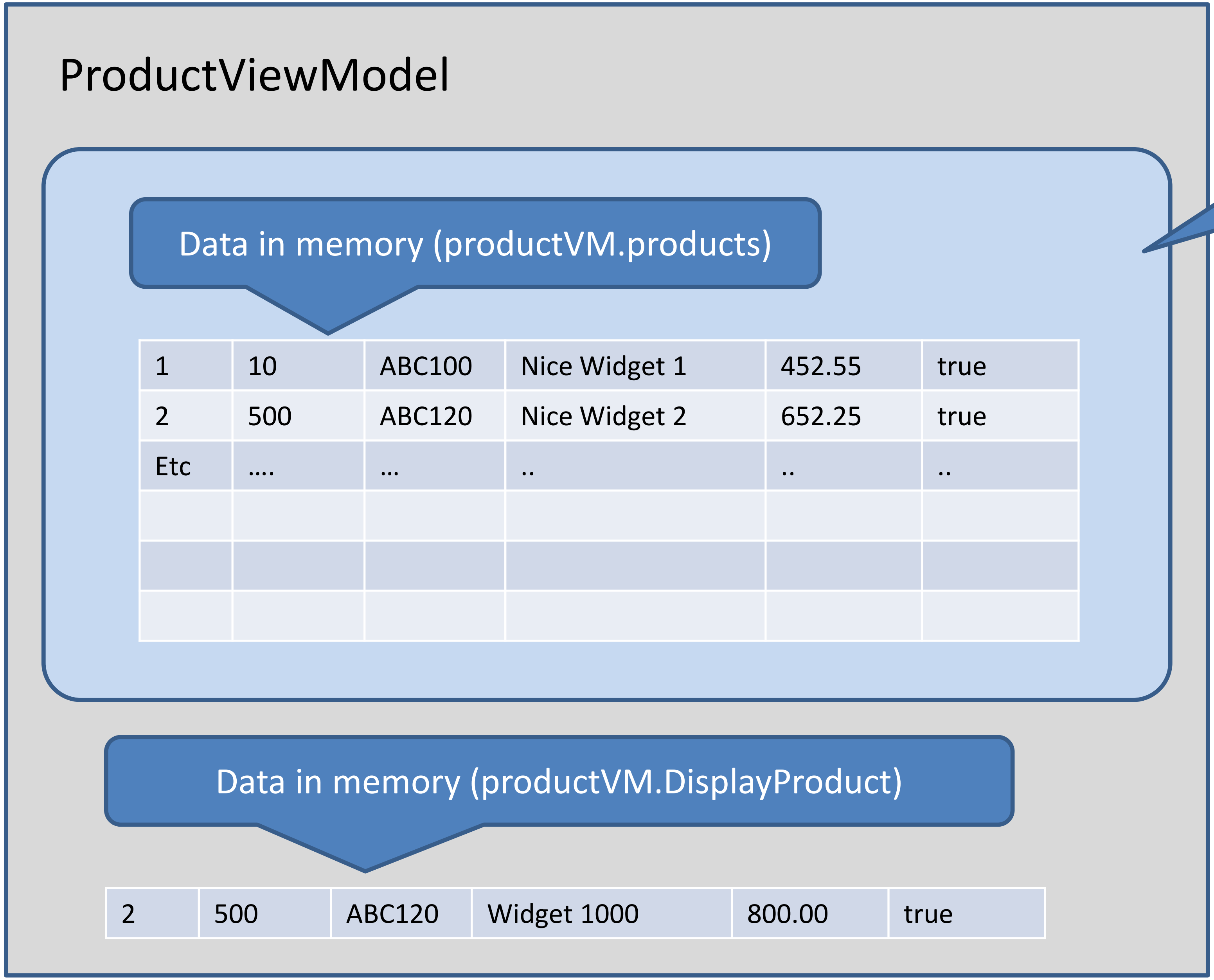
2	**500**	**ABC120**	**Nice Widget 1000**	**800.00**	**true**

ProductViewModel.products

This item is updated now after clicking Save

0	1	10	ABC100	Nice Widget 1	452.55	true
1	2	500	ABC120	Nice Widget 1000	800.00	true
2	Etc	….	…	..	..	..
3						
4						
5						

Product ViewModel explained



ProductViewModel

Data in memory (productVM.products)

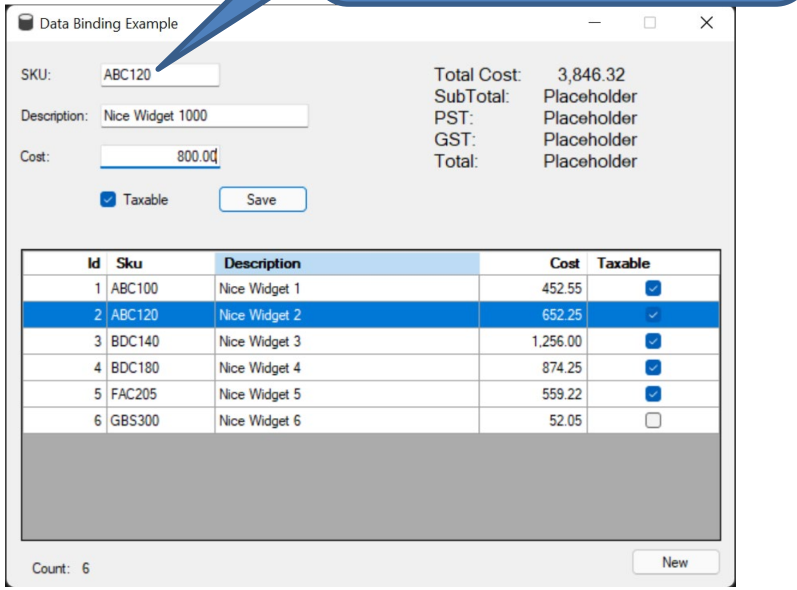
1	10	ABC100	Nice Widget 1	452.55	true
2	500	ABC120	Nice Widget 2	652.25	true
Etc	….	…	..	..	..

Data in memory (productVM.DisplayProduct)

2	500	ABC120	Widget 1000	800.00	true

BindingSource wraps around the list and watches for changes (productVM.ProductsSource)

Data also in Control properties (eg textBoxSku.Text)



Data Binding Example

SKU: ABC120

Description: Nice Widget 1000

Cost: 800.00

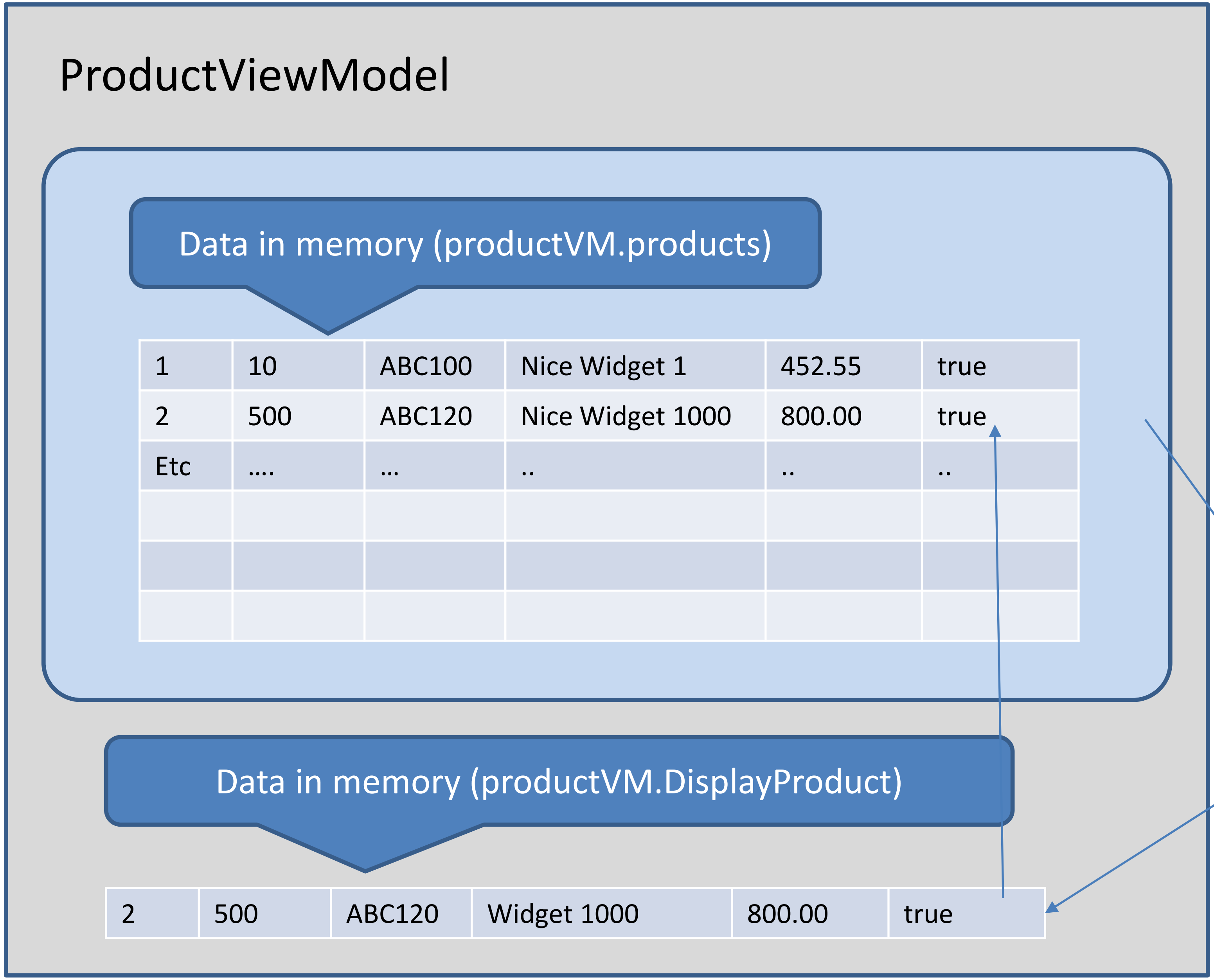
☑ Taxable

Total Cost: 3,846.32
SubTotal: Placeholder
PST: Placeholder
GST: Placeholder
Total: Placeholder

Id	Sku	Description	Cost	Taxable
1	ABC100	Nice Widget 1	452.55	☑
2	ABC120	Nice Widget 2	652.25	☑
3	BDC140	Nice Widget 3	1,256.00	☑
4	BDC180	Nice Widget 4	874.25	☑
5	FAC205	Nice Widget 5	559.22	☑
6	GBS300	Nice Widget 6	52.05	☐

Count: 6

Product ViewModel explained



ProductViewModel

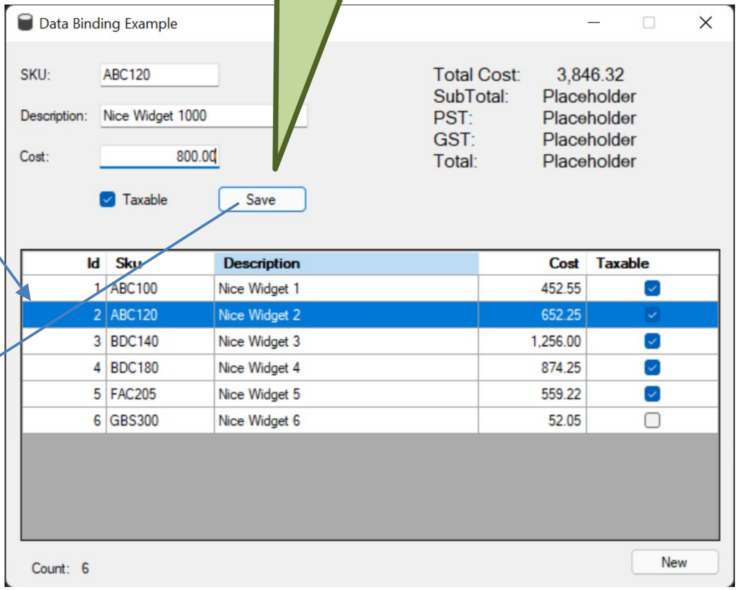
Data in memory (productVM.products)

1	10	ABC100	Nice Widget 1	452.55	true
2	500	ABC120	Nice Widget 1000	800.00	true
Etc	….	…	..	..	..

Data in memory (productVM.DisplayProduct)

2	500	ABC120	Widget 1000	800.00	true

Hitting the Save button causes the DisplayProduct to overwrite the item in the list, then an event is raised to inform the grid to refresh



SKU: ABC120

Description: Nice Widget 1000

Cost: 800.00

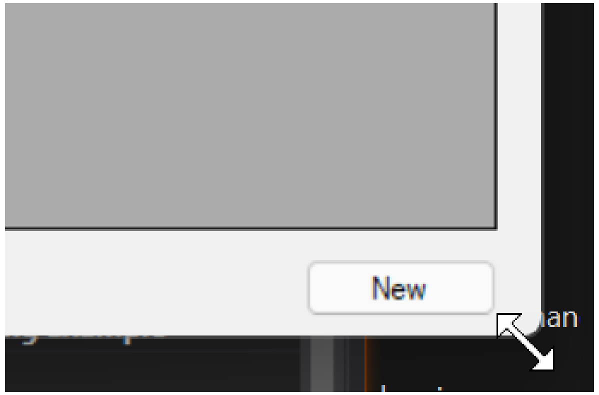
☑ Taxable

Total Cost: 3,846.32
SubTotal: Placeholder
PST: Placeholder
GST: Placeholder
Total: Placeholder

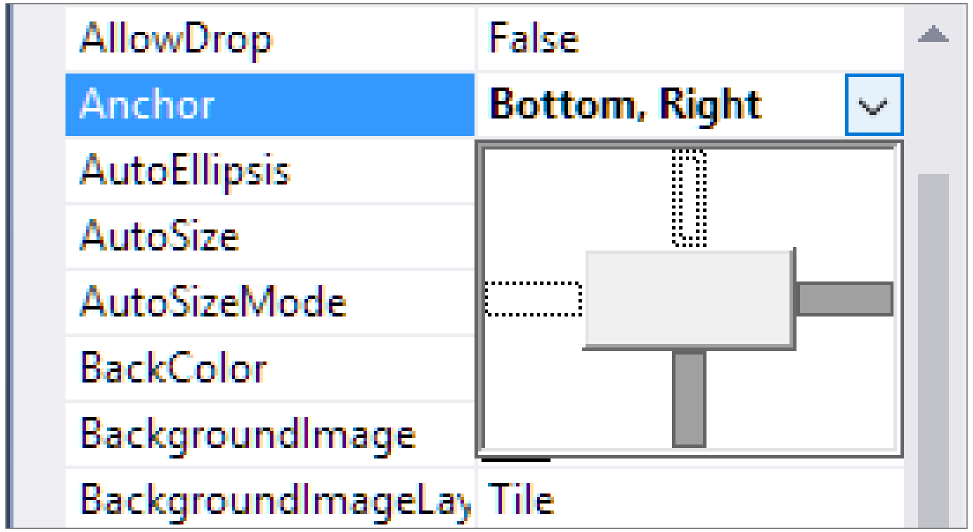
Id	Sku	Description	Cost	Taxable
1	ABC100	Nice Widget 1	452.55	☑
2	ABC120	Nice Widget 2	652.25	☑
3	BDC140	Nice Widget 3	1,256.00	☑
4	BDC180	Nice Widget 4	874.25	☑
5	FAC205	Nice Widget 5	559.22	☑
6	GBS300	Nice Widget 6	52.05	□

Count: 6

Forms– Button Anchoring and Form Resizing



Change Button anchoring from Top Left, to Bottom Right



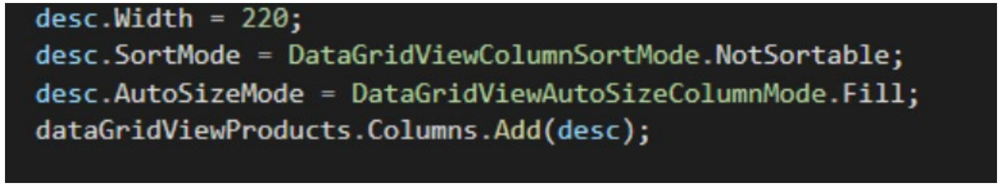
AllowDrop	False
Anchor	Bottom, Right
AutoEllipsis	
AutoSize	
AutoSizeMode	
BackColor	
BackgroundImage	
BackgroundImageLay	Tile

Change the anchoring of the Newbutton to Bottom, Right. (default is Top, Left)

Both buttons will then track as you adjust the form's size in design mode and when the form is resized while the application is running.

Set the description column to autosize and expand/shrink when the grid is resized.

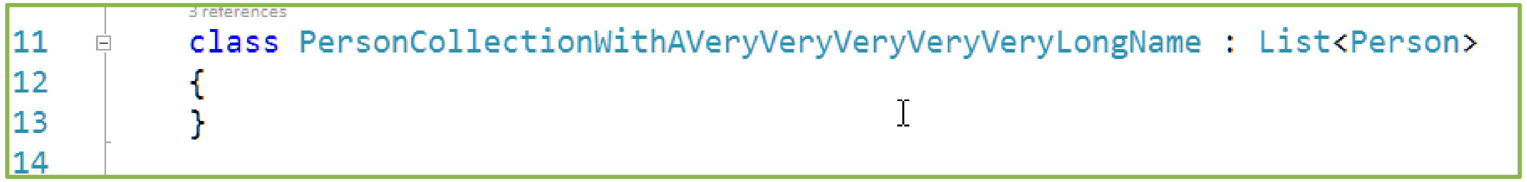
Don't forget to set a minimum size for the form!



```
desc.Width = 220;
desc.SortMode = DataGridViewColumnSortMode.NotSortable;
desc.AutoSizeMode = DataGridViewAutoSizeColumnMode.Fill;
dataGridViewProducts.Columns.Add(desc);
```

Type Inference

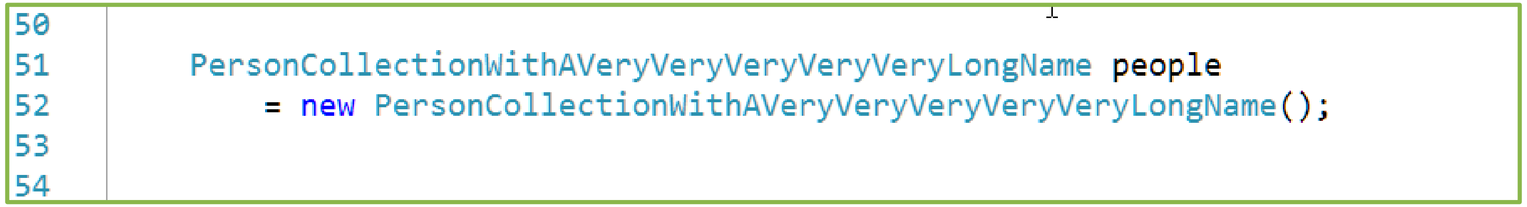
Collection Class Inherited From List of Type Person



3 references

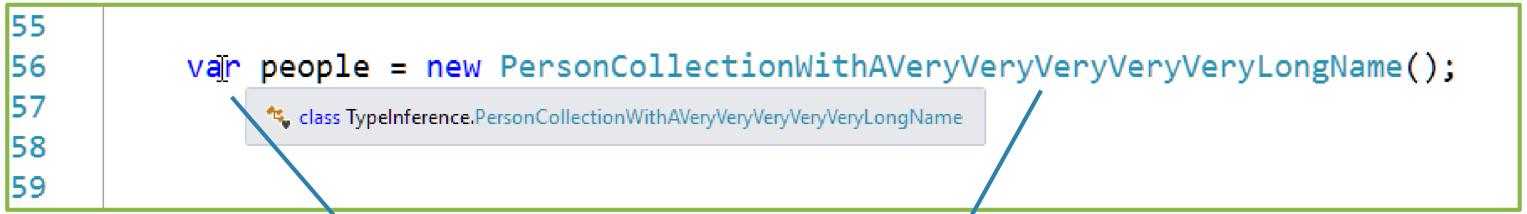
```
class PersonCollectionWithAVeryVeryVeryVeryVeryLongName : List<Person>
{
}
```

Conventional Assignment Statement



```
PersonCollectionWithAVeryVeryVeryVeryVeryLongName people
    = new PersonCollectionWithAVeryVeryVeryVeryVeryLongName();
```

Can Be Rewritten as ...



```
var people = new PersonCollectionWithAVeryVeryVeryVeryVeryLongName();
```

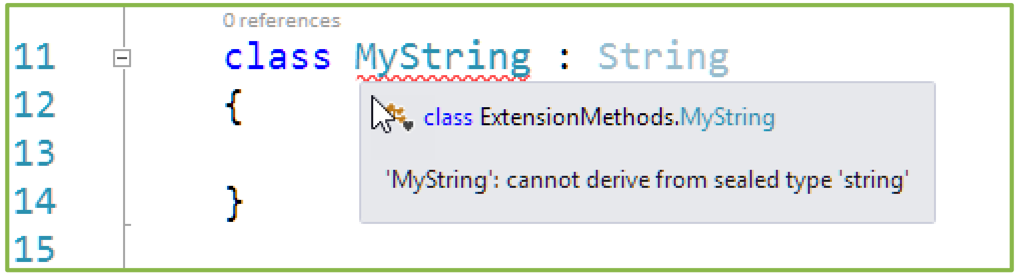
class TypeInference.PersonCollectionWithAVeryVeryVeryVeryVeryLongName

Type is inferred from RHS

- Can specify the var keyword on the LHS of an assignment statement in place of the actual data type
- Compiler infers the data type from the RHS of the assignment
- Can be used for local variables only
- Can not be used for method parameter or return types
- Can not be used for fields (instance variables)
- Variable declaration and assignment must occur in a single statement

Extension Methods

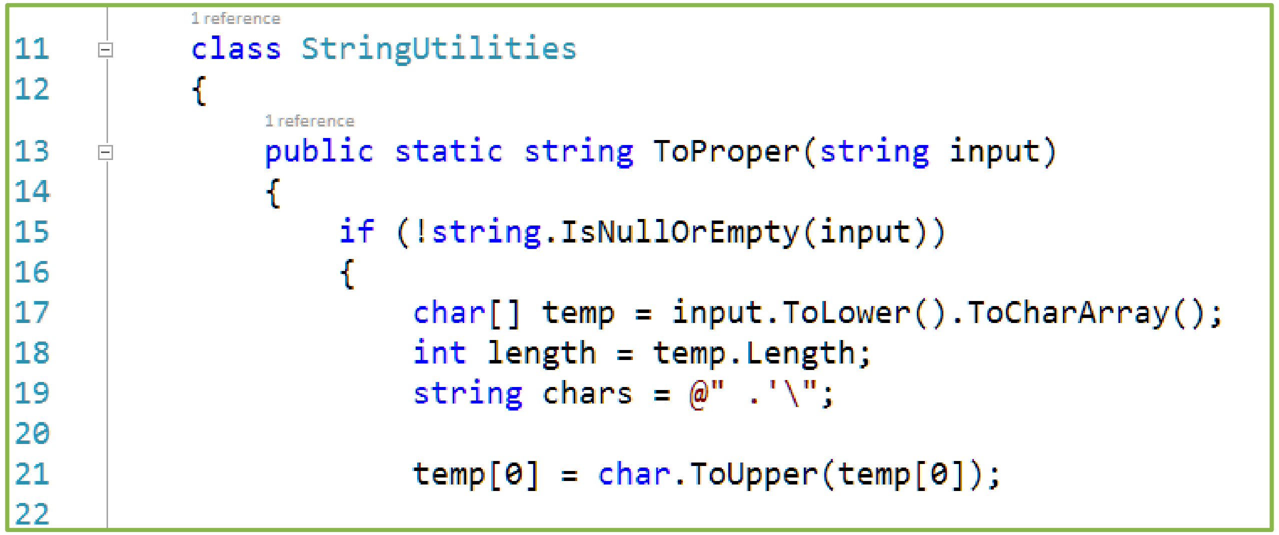
Cannot extend the string class because it is sealed



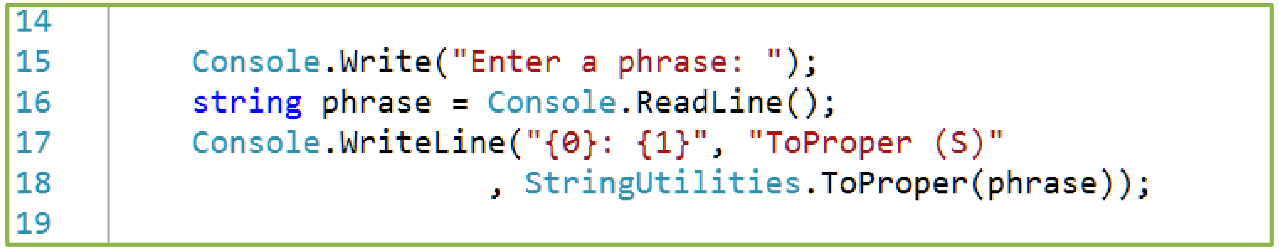
```
0 references
class MyString : String
{

}
```

class ExtensionMethods.MyString

'MyString': cannot derive from sealed type 'string'



```
1 reference
class StringUtilities
{
    1 reference
    public static string ToProper(string input)
    {
        if (!string.IsNullOrEmpty(input))
        {
            char[] temp = input.ToLower().ToCharArray();
            int length = temp.Length;
            string chars = @" .'\";

            temp[0] = char.ToUpper(temp[0]);

```

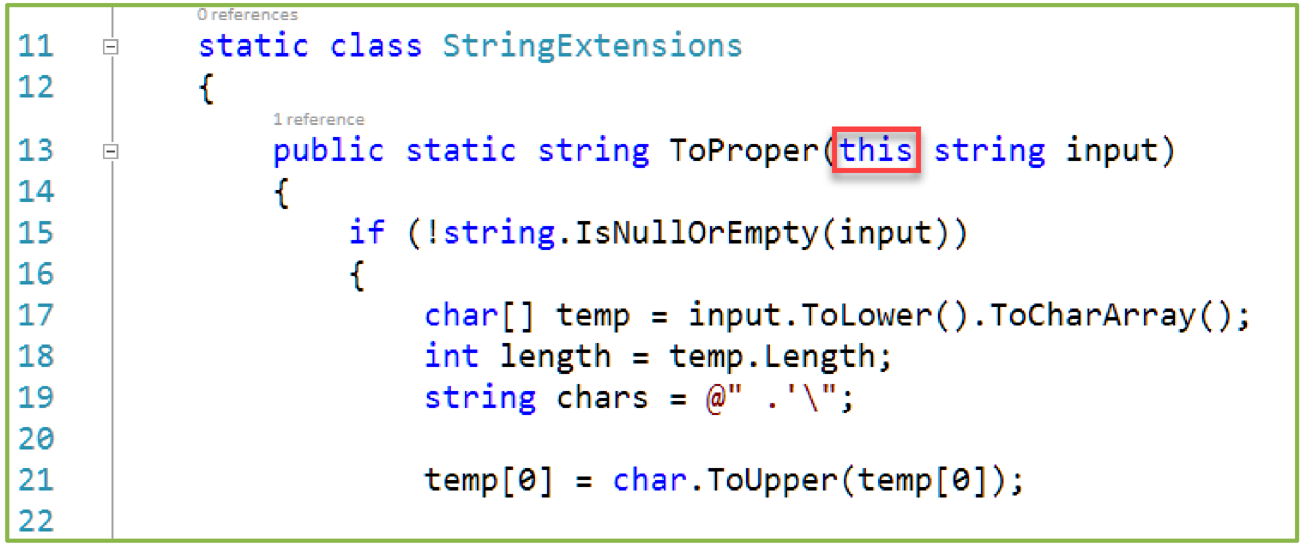


```

Console.Write("Enter a phrase: ");
string phrase = Console.ReadLine();
Console.WriteLine("{0}: {1}", "ToProper (S)"
                  , StringUtilities.ToProper(phrase));

```

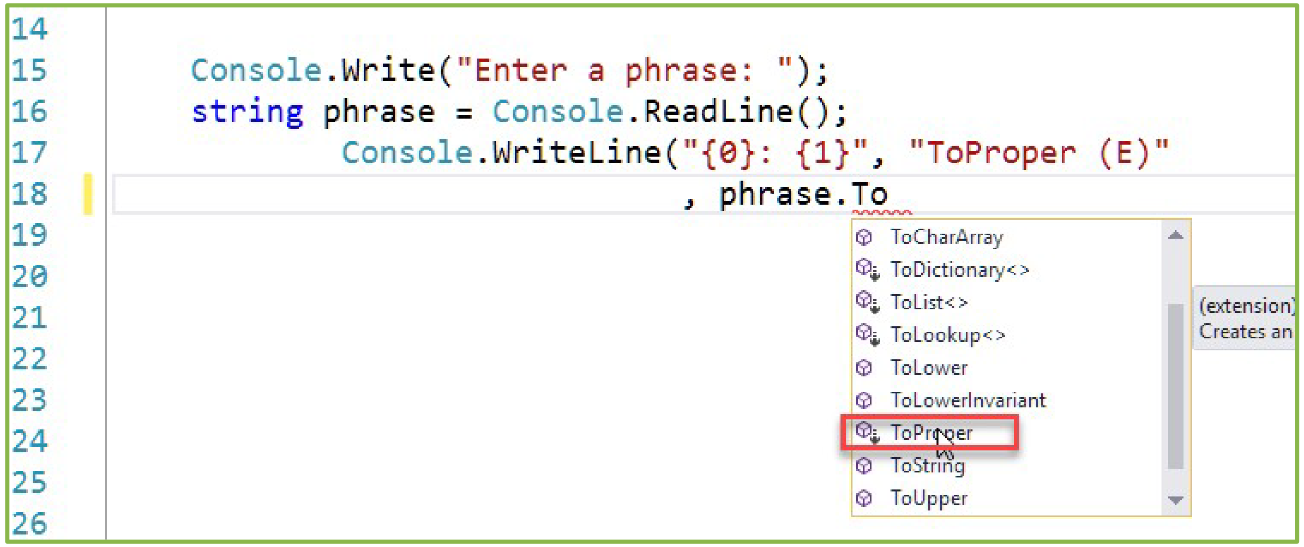
Static Method

Normal solution would be to write a static method to provide the desired functionality

Extension Methods



```
static class StringExtensions
{
    public static string ToProper(this string input)
    {
        if (!string.IsNullOrEmpty(input))
        {
            char[] temp = input.ToLower().ToCharArray();
            int length = temp.Length;
            string chars = @" .'\";

            temp[0] = char.ToUpper(temp[0]);

```



```

Console.Write("Enter a phrase: ");
string phrase = Console.ReadLine();
        Console.WriteLine("{0}: {1}", "ToProper (E)"
                        , phrase.To
```

- A means of seemingly adding functionality to a sealed class
- Static method created in a static class
- Disguises a static method to appear as an instance method (of the pseudo extended type)
- Data type of first parameter is type that is extended
- First parameter is defined with the ‘this’ keyword
- Can have multiple parameters
- Method only has access to the public members of the ‘extended’ type
- Must include class namespace (if different)
- Can also be invoked like a normal static method

LINQ – Language Integrated Query

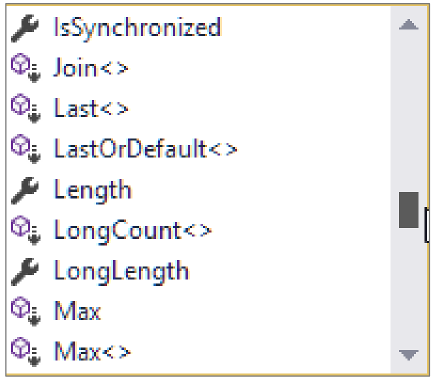
```
select col1, col2  from table1
where colx = condition
```

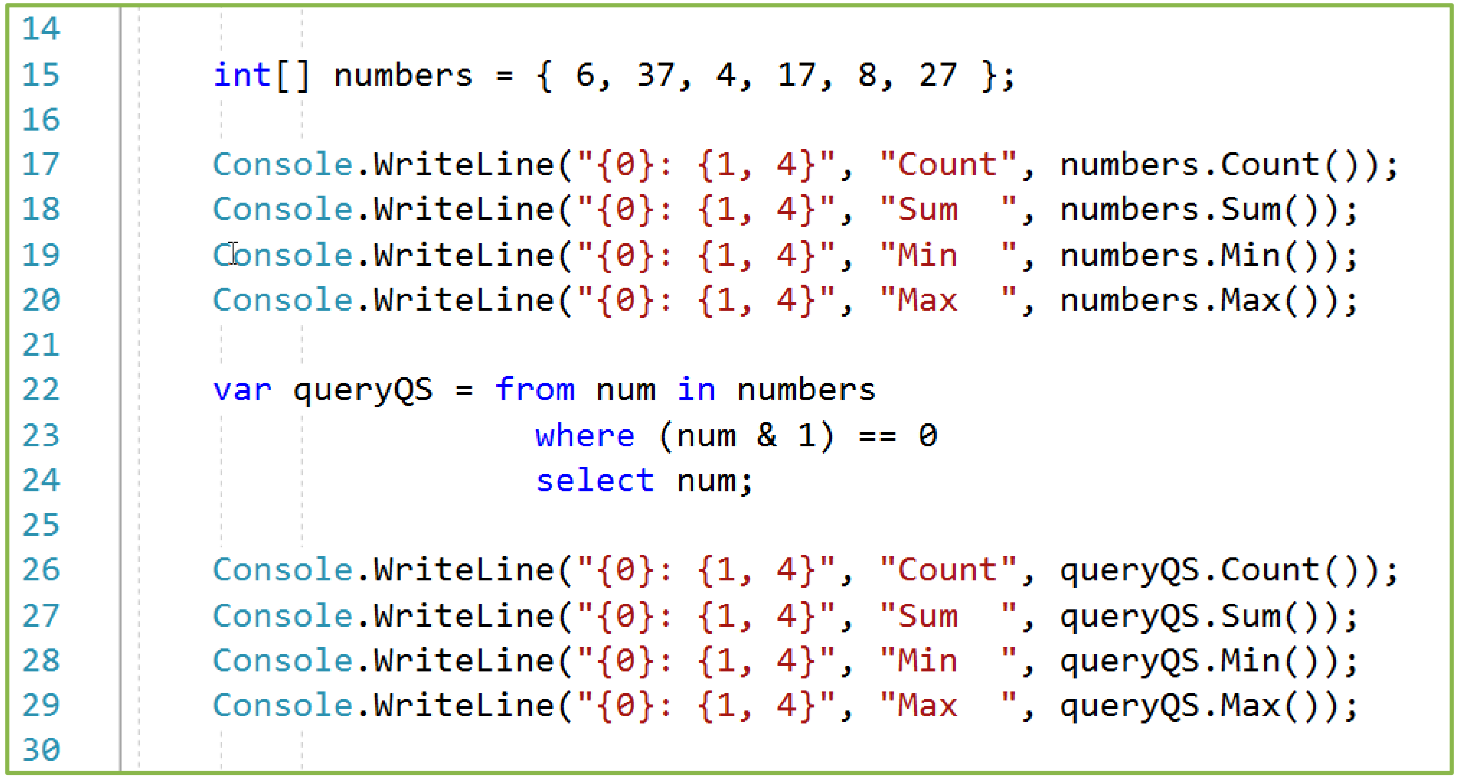
LINQ

```
from product in products where product.Taxable == true  select
product.Sku, product.Price
```

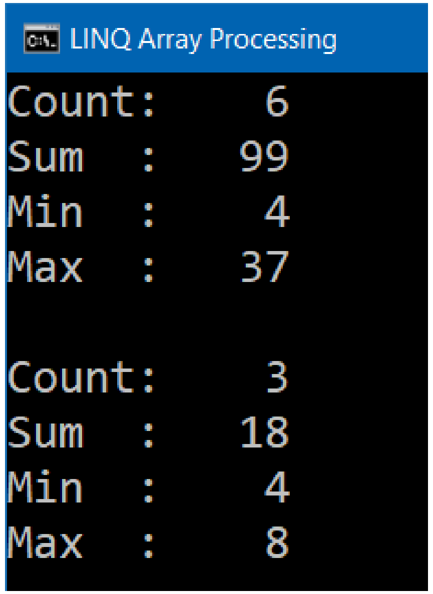
LINQ - Arrays

LINQ defines several Extension Methods on the IEnumerable<T> Interface





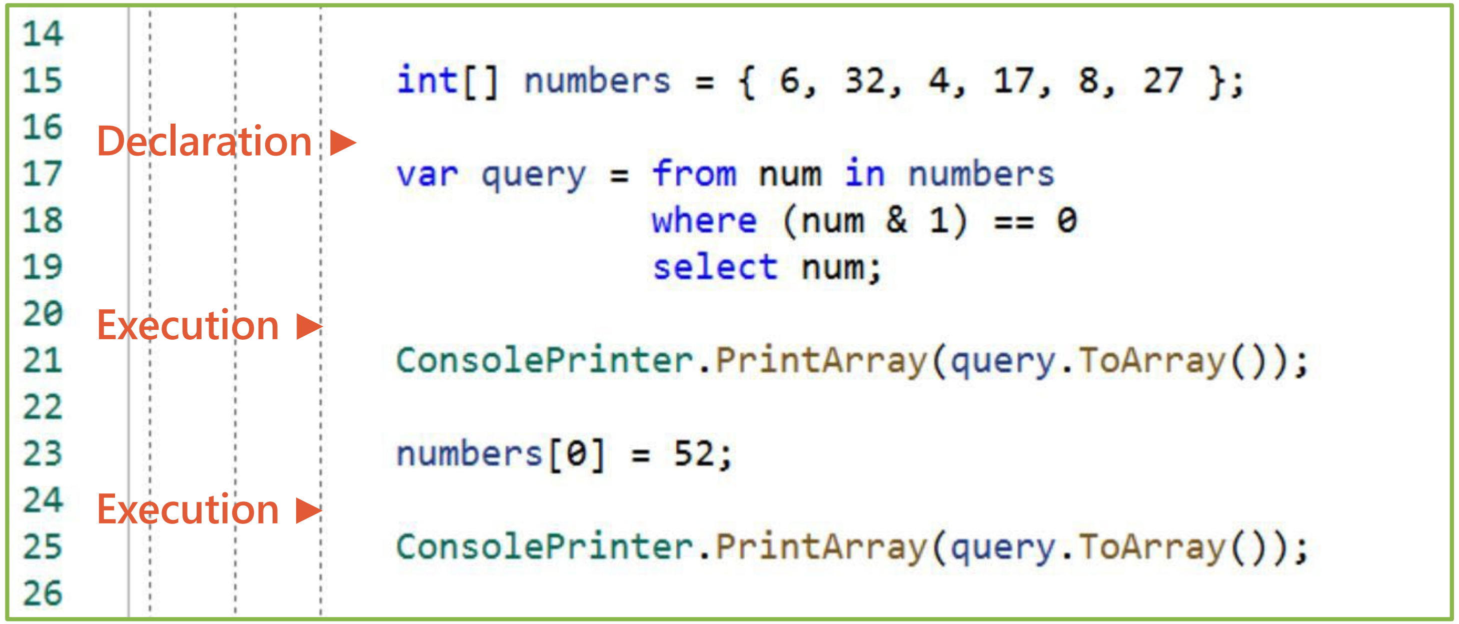
```
int[] numbers = { 6, 37, 4, 17, 8, 27 };

Console.WriteLine("{0}: {1, 4}", "Count", numbers.Count());
Console.WriteLine("{0}: {1, 4}", "Sum  ", numbers.Sum());
Console.WriteLine("{0}: {1, 4}", "Min  ", numbers.Min());
Console.WriteLine("{0}: {1, 4}", "Max  ", numbers.Max());

var queryQS = from num in numbers
                where (num & 1) == 0
                select num;

Console.WriteLine("{0}: {1, 4}", "Count", queryQS.Count());
Console.WriteLine("{0}: {1, 4}", "Sum  ", queryQS.Sum());
Console.WriteLine("{0}: {1, 4}", "Min  ", queryQS.Min());
Console.WriteLine("{0}: {1, 4}", "Max  ", queryQS.Max());
```



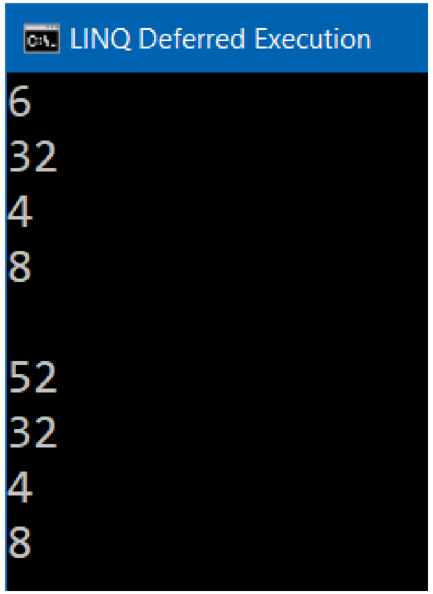
```
Count:    6
Sum  :   99
Min  :    4
Max  :   37

Count:    3
Sum  :   18
Min  :    4
Max  :    8
```

LINQ – Deferred Execution

The query does not execute when it is declared – it will execute when an operation is called on it such as ToArray(), ToList() or is enumerated in a loop

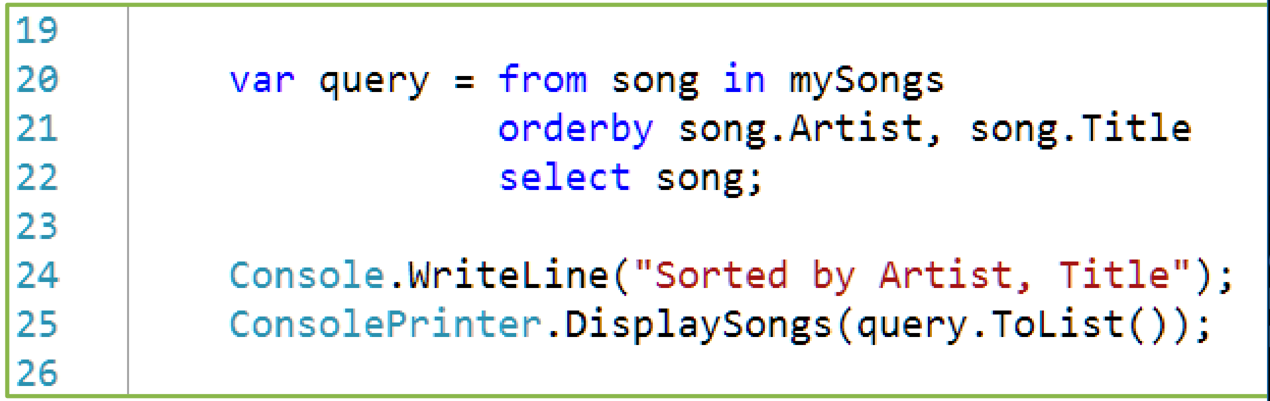


```
int[] numbers = { 6, 32, 4, 17, 8, 27 };

var query = from num in numbers
            where (num & 1) == 0
            select num;

ConsolePrinter.PrintArray(query.ToArray());

numbers[0] = 52;

ConsolePrinter.PrintArray(query.ToArray());
```

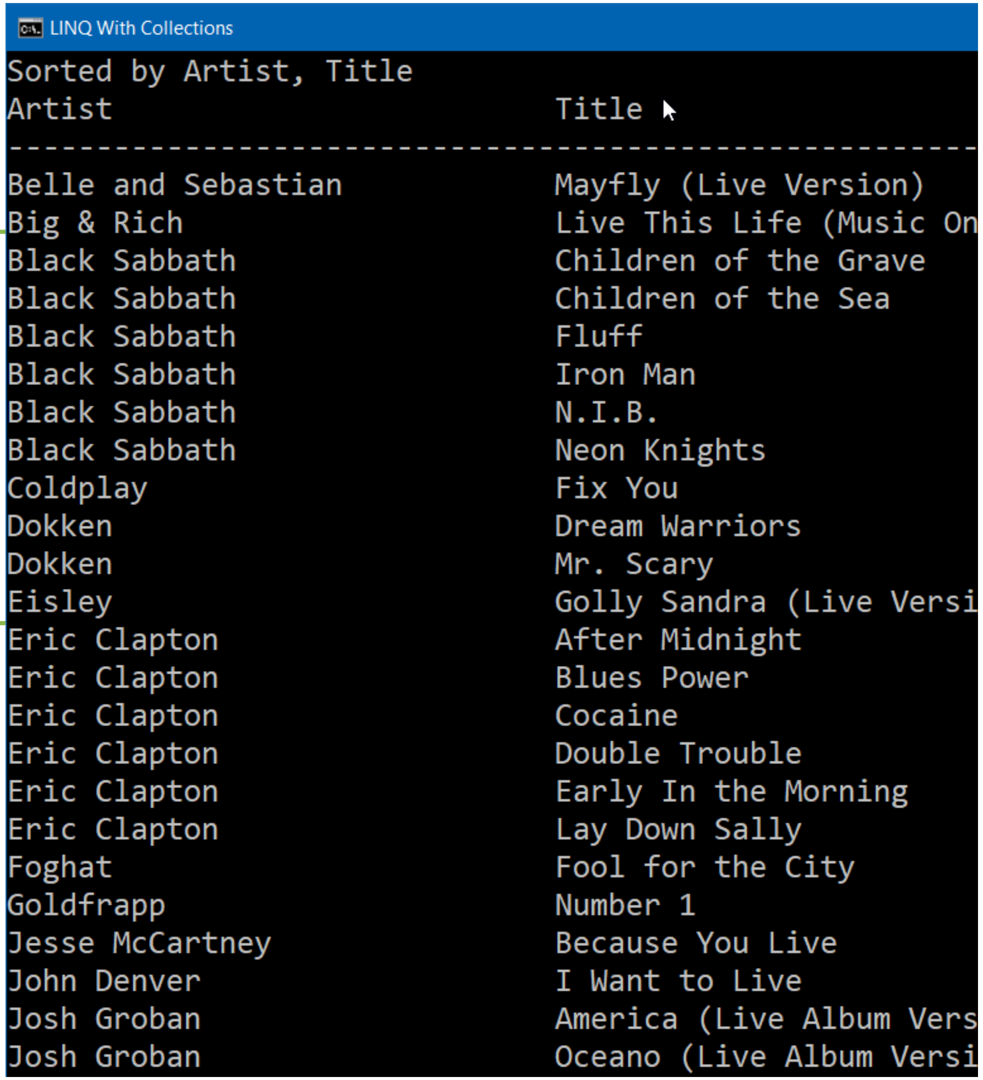


```
6
32
4
8

52
32
4
8
```

LINQ – Collection Queries

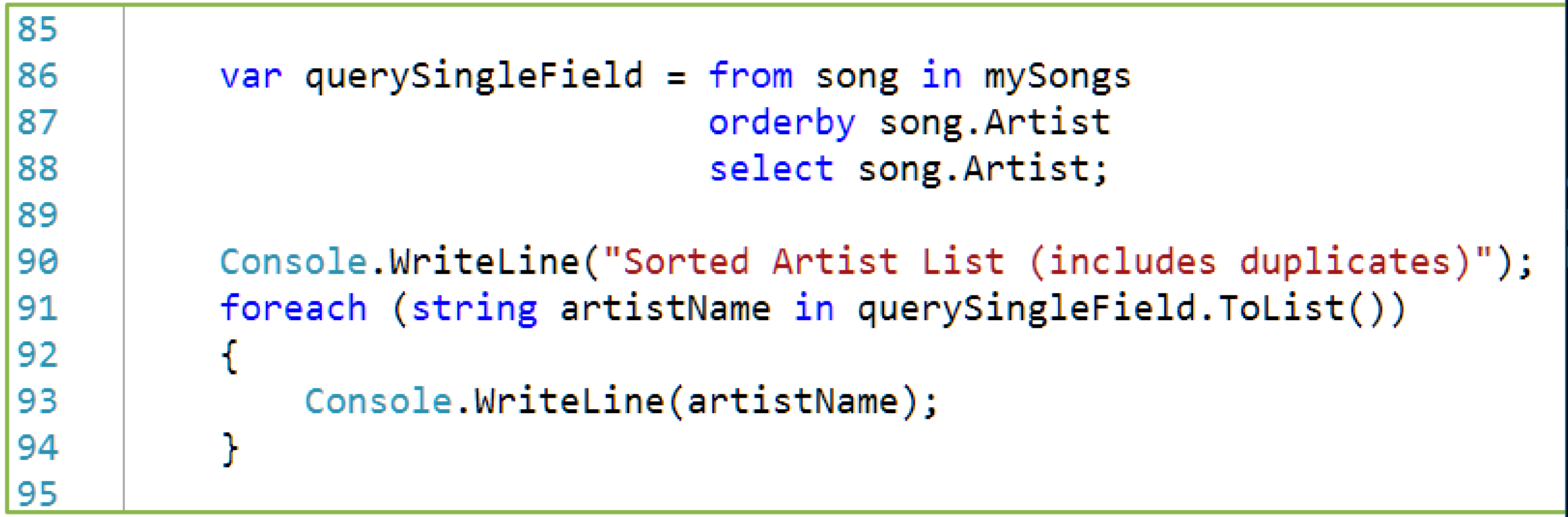


```
        var query = from song in mySongs
                    orderby song.Artist, song.Title
                    select song;

        Console.WriteLine("Sorted by Artist, Title");
        ConsolePrinter.DisplaySongs(query.ToList());
```

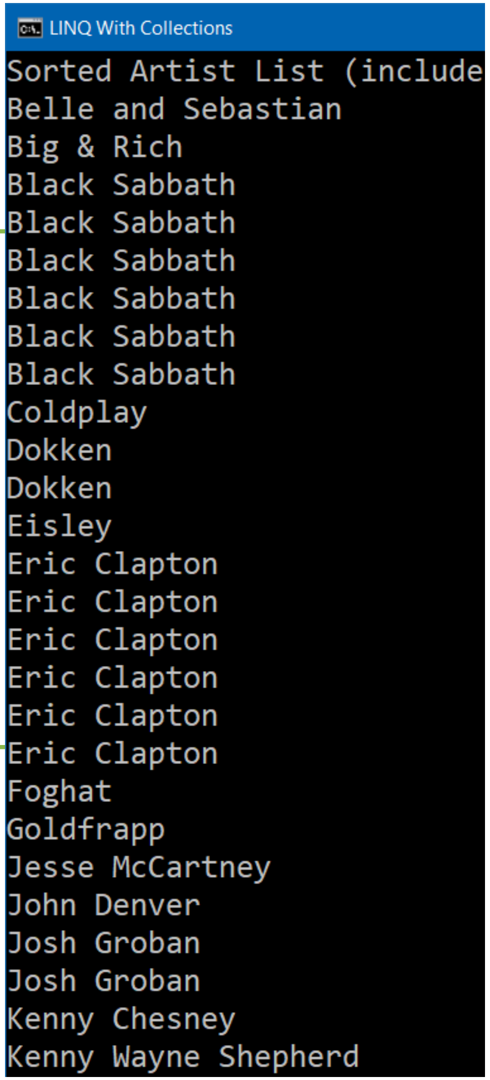


```
Sorted by Artist, Title
Artist                         Title
-------------------------------------------------------
Belle and Sebastian            Mayfly (Live Version)
Big & Rich                     Live This Life (Music On
Black Sabbath                  Children of the Grave
Black Sabbath                  Children of the Sea
Black Sabbath                  Fluff
Black Sabbath                  Iron Man
Black Sabbath                  N.I.B.
Black Sabbath                  Neon Knights
Coldplay                       Fix You
Dokken                         Dream Warriors
Dokken                         Mr. Scary
Eisley                         Golly Sandra (Live Versi
Eric Clapton                   After Midnight
Eric Clapton                   Blues Power
Eric Clapton                   Cocaine
Eric Clapton                   Double Trouble
Eric Clapton                   Early In the Morning
Eric Clapton                   Lay Down Sally
Foghat                         Fool for the City
Goldfrapp                      Number 1
Jesse McCartney                Because You Live
John Denver                    I Want to Live
Josh Groban                    America (Live Album Vers
Josh Groban                    Oceano (Live Album Versi
```

LINQ – Collection Queries

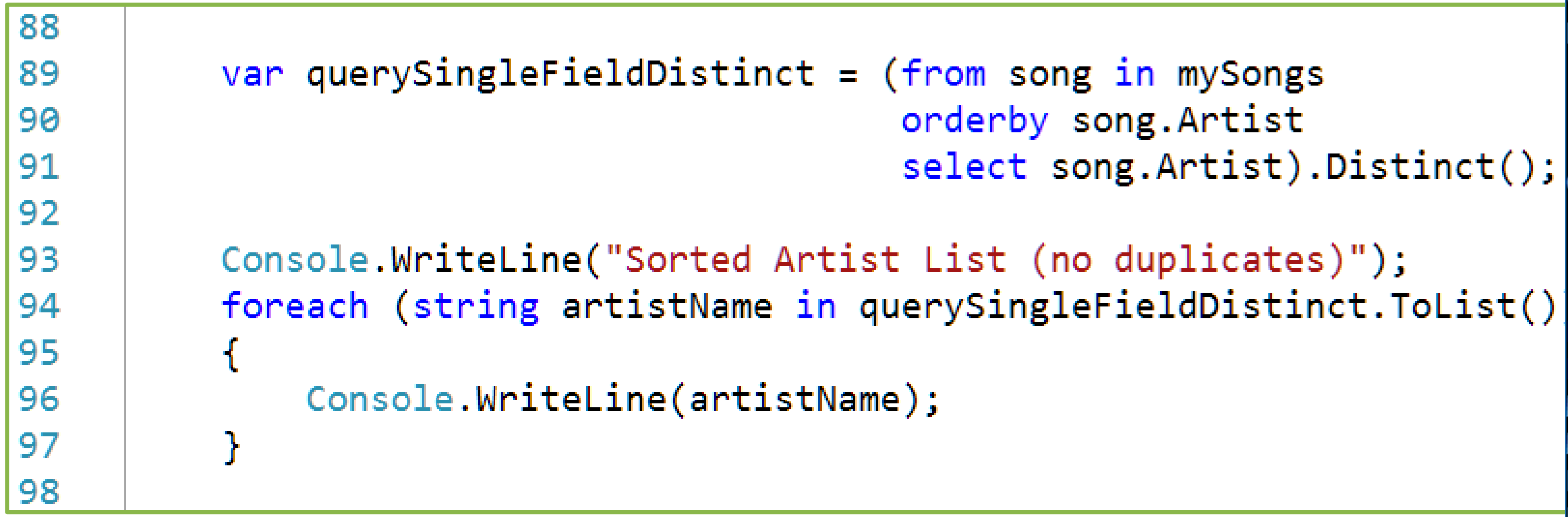


```
var querySingleField = from song in mySongs
                       orderby song.Artist
                       select song.Artist;

Console.WriteLine("Sorted Artist List (includes duplicates)");
foreach (string artistName in querySingleField.ToList())
{
    Console.WriteLine(artistName);
}
```

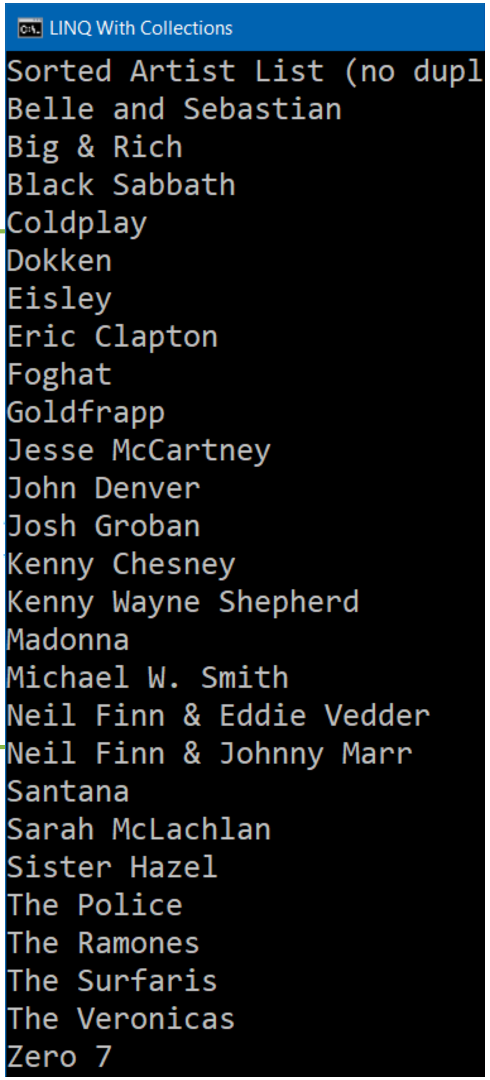


```
Sorted Artist List (include
Belle and Sebastian
Big & Rich
Black Sabbath
Black Sabbath
Black Sabbath
Black Sabbath
Black Sabbath
Black Sabbath
Coldplay
Dokken
Dokken
Eisley
Eric Clapton
Eric Clapton
Eric Clapton
Eric Clapton
Eric Clapton
Eric Clapton
Foghat
Goldfrapp
Jesse McCartney
John Denver
Josh Groban
Josh Groban
Kenny Chesney
Kenny Wayne Shepherd
```

LINQ – Collection Queries

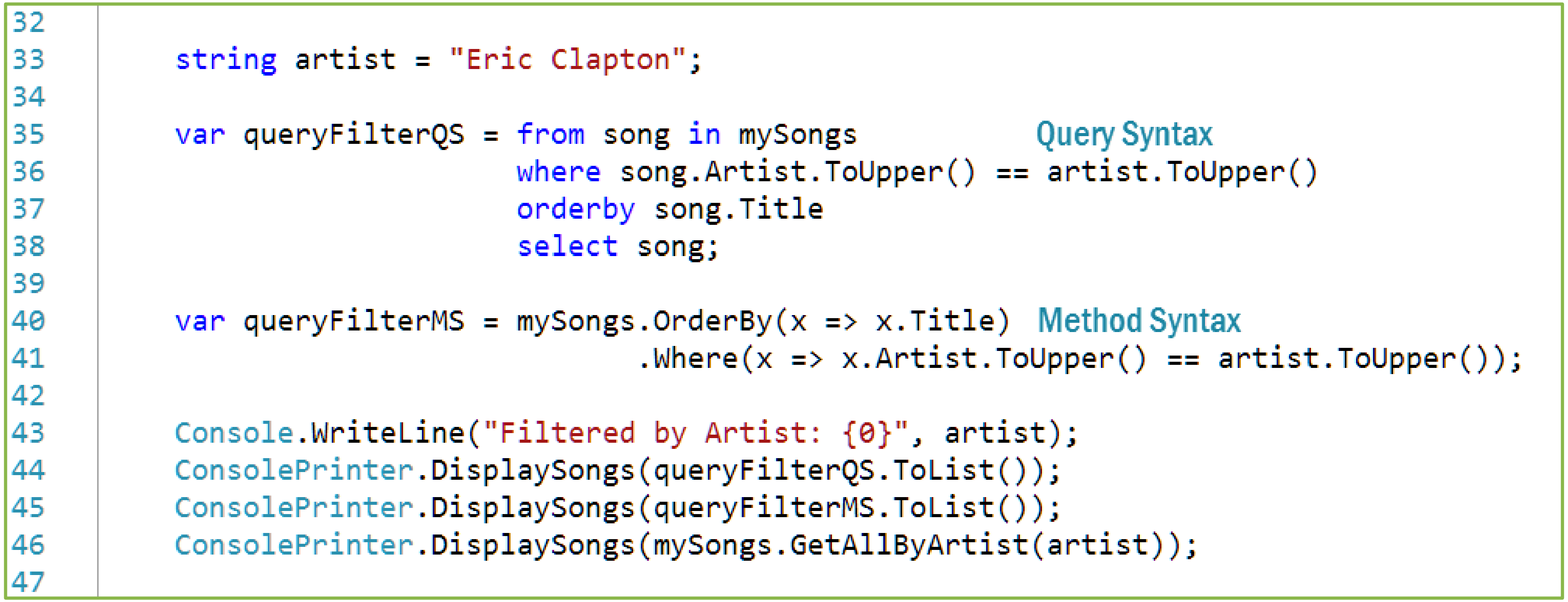


```

    var querySingleFieldDistinct = (from song in mySongs
                                    orderby song.Artist
                                    select song.Artist).Distinct();

    Console.WriteLine("Sorted Artist List (no duplicates)");
    foreach (string artistName in querySingleFieldDistinct.ToList()
    {
        Console.WriteLine(artistName);
    }

```



```
Sorted Artist List (no dupl
Belle and Sebastian
Big & Rich
Black Sabbath
Coldplay
Dokken
Eisley
Eric Clapton
Foghat
Goldfrapp
Jesse McCartney
John Denver
Josh Groban
Kenny Chesney
Kenny Wayne Shepherd
Madonna
Michael W. Smith
Neil Finn & Eddie Vedder
Neil Finn & Johnny Marr
Santana
Sarah McLachlan
Sister Hazel
The Police
The Ramones
The Surfaris
The Veronicas
Zero 7
```

LINQ – Query vs Method Syntax



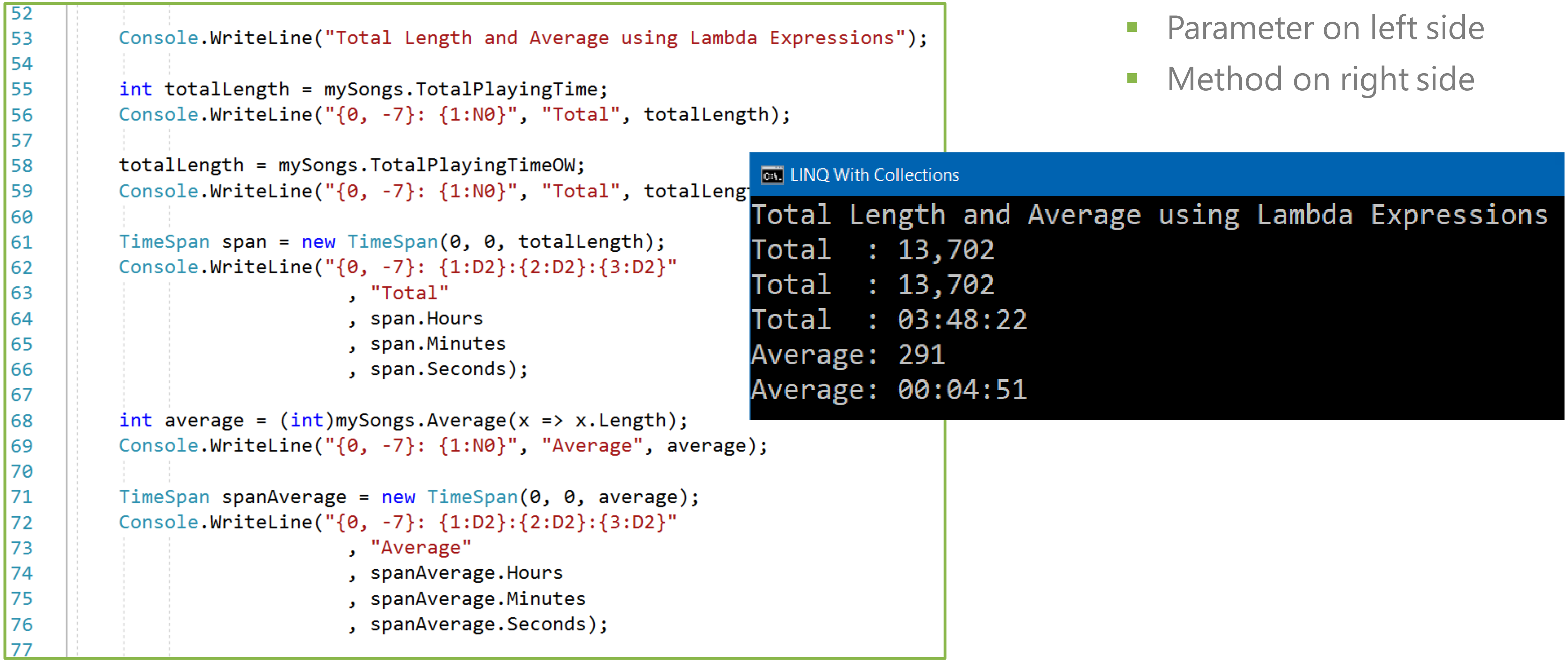
```

    string artist = "Eric Clapton";

    var queryFilterQS = from song in mySongs
                        where song.Artist.ToUpper() == artist.ToUpper()
                        orderby song.Title
                        select song;

    var queryFilterMS = mySongs.OrderBy(x => x.Title)
                               .Where(x => x.Artist.ToUpper() == artist.ToUpper());

    Console.WriteLine("Filtered by Artist: {0}", artist);
    ConsolePrinter.DisplaySongs(queryFilterQS.ToList());
    ConsolePrinter.DisplaySongs(queryFilterMS.ToList());
    ConsolePrinter.DisplaySongs(mySongs.GetAllByArtist(artist));

```

Lambda Expressions

```
 (param => method)
Sum(x => x.Length)
```

x "goes to" x.Length

- Anonymous inline method
- => "goes to" operator
- Parameter on left side
- Method on right side

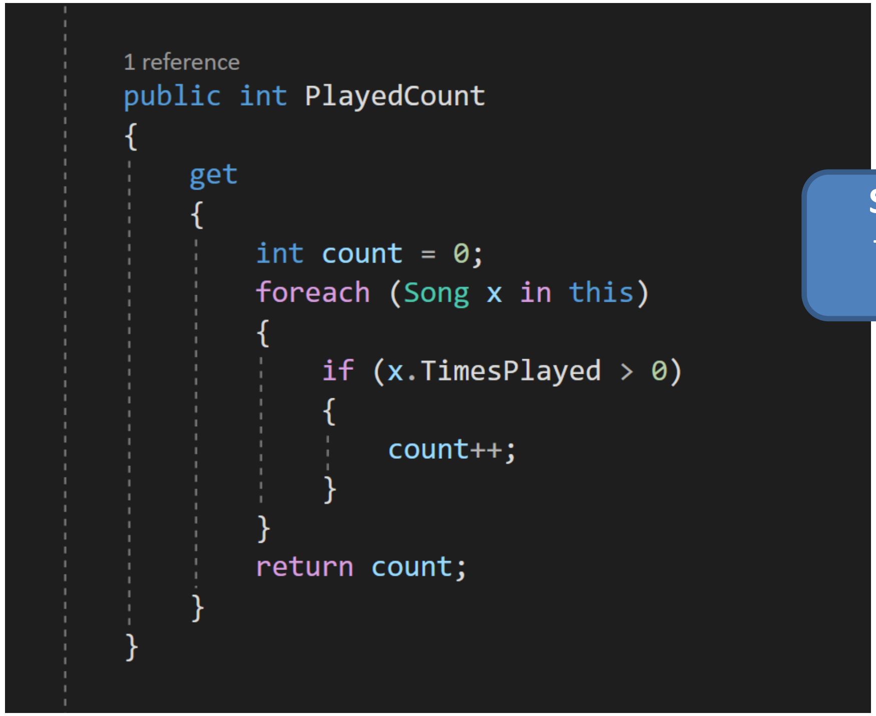


```
Console.WriteLine("Total Length and Average using Lambda Expressions");

int totalLength = mySongs.TotalPlayingTime;
Console.WriteLine("{0, -7}: {1:N0}", "Total", totalLength);

totalLength = mySongs.TotalPlayingTimeOW;
Console.WriteLine("{0, -7}: {1:N0}", "Total", totalLeng

TimeSpan span = new TimeSpan(0, 0, totalLength);
Console.WriteLine("{0, -7}: {1:D2}:{2:D2}:{3:D2}"
                    , "Total"
                    , span.Hours
                    , span.Minutes
                    , span.Seconds);

int average = (int)mySongs.Average(x => x.Length);
Console.WriteLine("{0, -7}: {1:N0}", "Average", average);

TimeSpan spanAverage = new TimeSpan(0, 0, average);
Console.WriteLine("{0, -7}: {1:D2}:{2:D2}:{3:D2}"
                    , "Average"
                    , spanAverage.Hours
                    , spanAverage.Minutes
                    , spanAverage.Seconds);
```

```
Total Length and Average using Lambda Expressions
Total  : 13,702
Total  : 13,702
Total  : 03:48:22
Average: 291
Average: 00:04:51
```

Lambda Expressions

Calculated Property



```
1 reference
public int PlayedCount
{
    get
    {
        int count = 0;
        foreach (Song x in this)
        {
            if (x.TimesPlayed > 0)
            {
                count++;
            }
        }
        return count;
    }
}
```

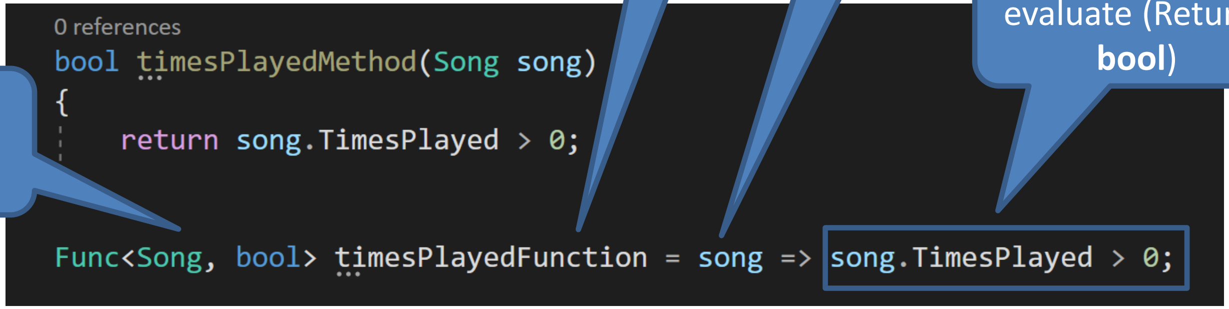
Song is input type, **bool** is output type

Method vs Func<t> delegate

Input parameter (name song, type **Song**)

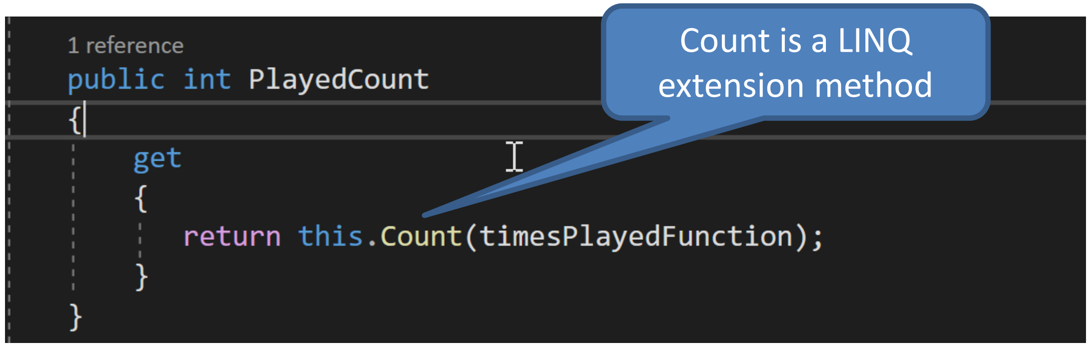
Reference label

Method to evaluate (Returns **bool**)



```
0 references
bool timesPlayedMethod(Song song)
{
    return song.TimesPlayed > 0;
}

Func<Song, bool> timesPlayedFunction = song => song.TimesPlayed > 0;
```

Func<T> can be used to create a reference to a method



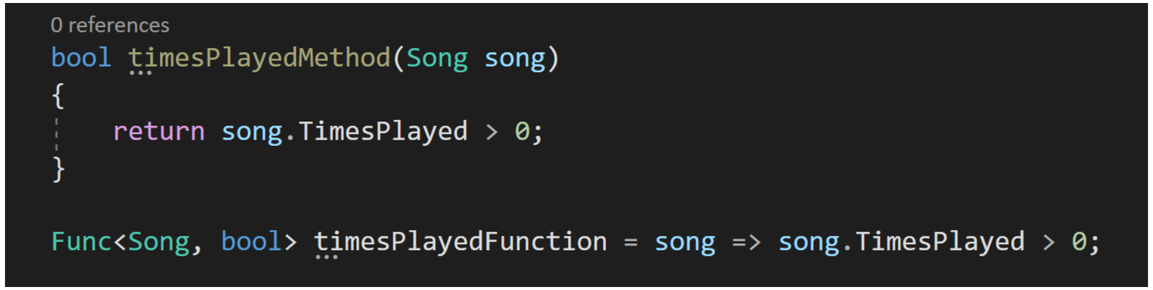
```
1 reference
public int PlayedCount
{
    get
    {
        return this.Count(timesPlayedFunction);
    }
}
```

Count is a LINQ extension method

We can then pass this reference in as a parameter to define a method that gets called on each item in a collection

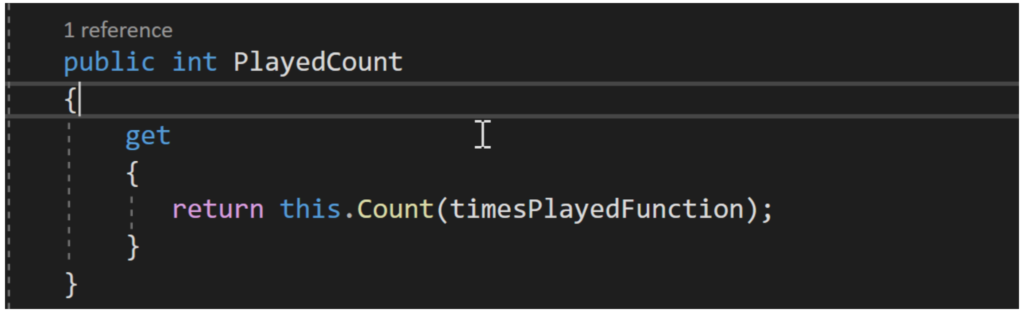
Lambda Expressions

Method vs Func<t> delegate



```
0 references
bool timesPlayedMethod(Song song)
{
    return song.TimesPlayed > 0;
}

Func<Song, bool> timesPlayedFunction = song => song.TimesPlayed > 0;
```

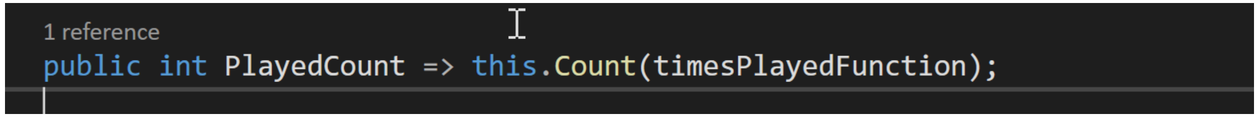
Func<T> can be used to create a reference to a method



```
1 reference
public int PlayedCount
{
    get
    {
        return this.Count(timesPlayedFunction);
    }
}
```

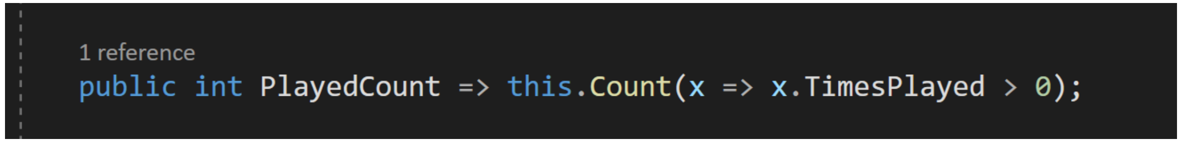
We can then pass this reference in as a parameter to define a method that gets called on each item in a collection

We can use the expression bodied style to save space



```
1 reference
public int PlayedCount => this.Count(timesPlayedFunction);
```

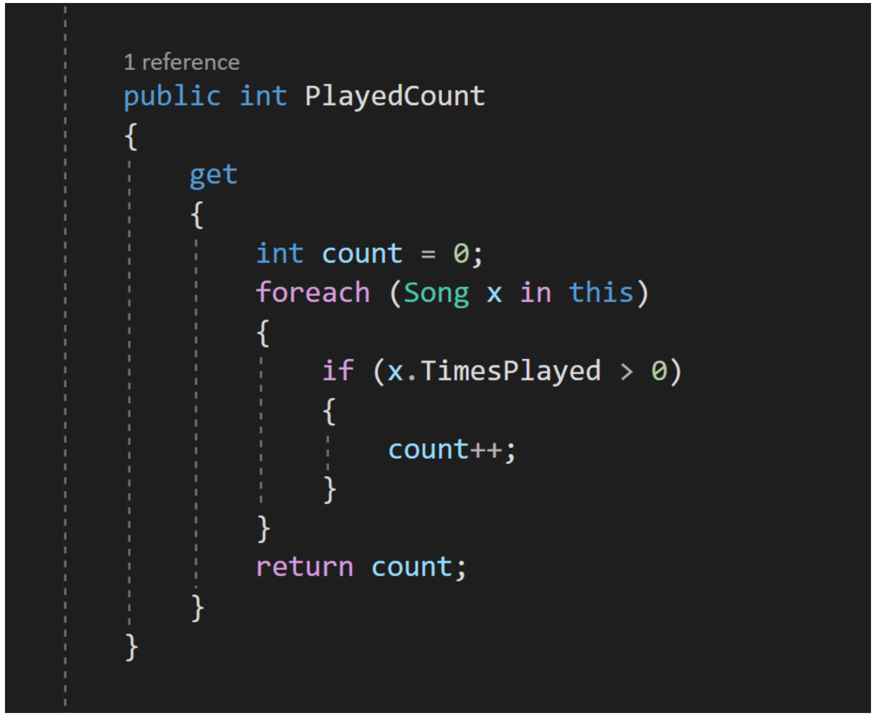
More commonly, instead of defining a Func<T> and referencing it, we just define it inline



```
1 reference
public int PlayedCount => this.Count(x => x.TimesPlayed > 0);
```

Lambda Expressions

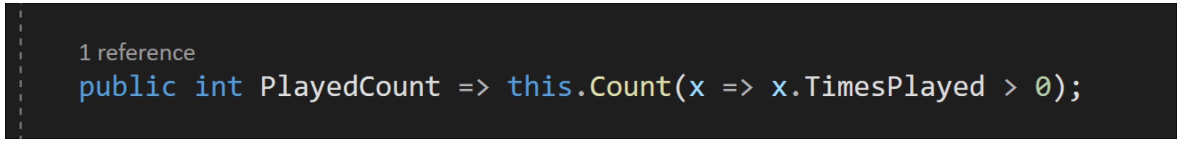
We end up going from this:



```
1 reference
public int PlayedCount
{
    get
    {
        int count = 0;
        foreach (Song x in this)
        {
            if (x.TimesPlayed > 0)
            {
                count++;
            }
        }
        return count;
    }
}
```

Calculated property

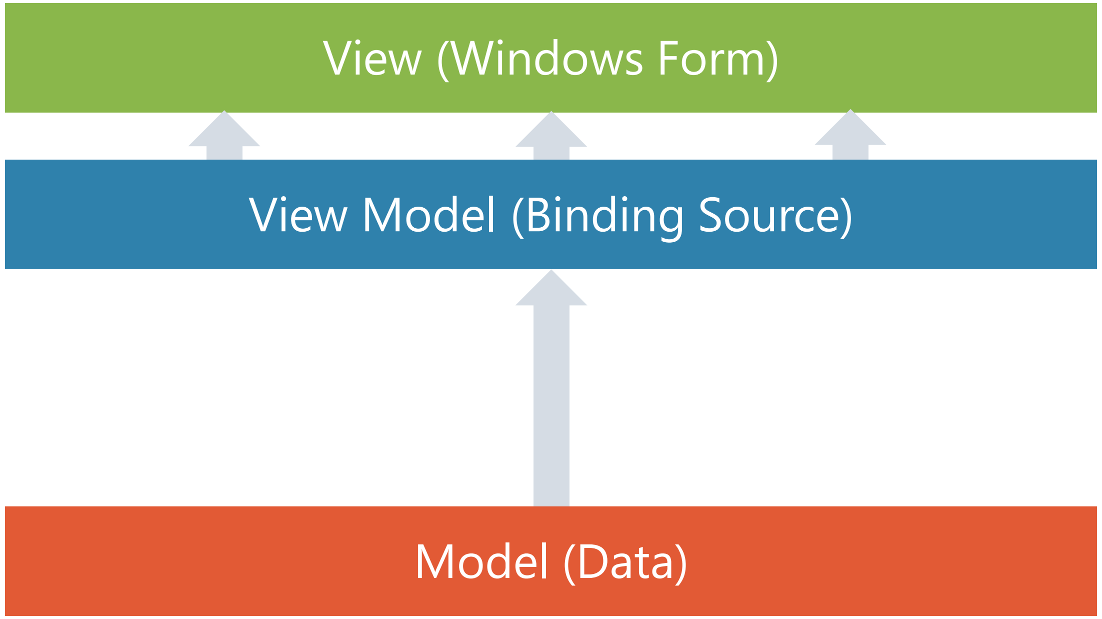
To this:



```
1 reference
public int PlayedCount => this.Count(x => x.TimesPlayed > 0);
```

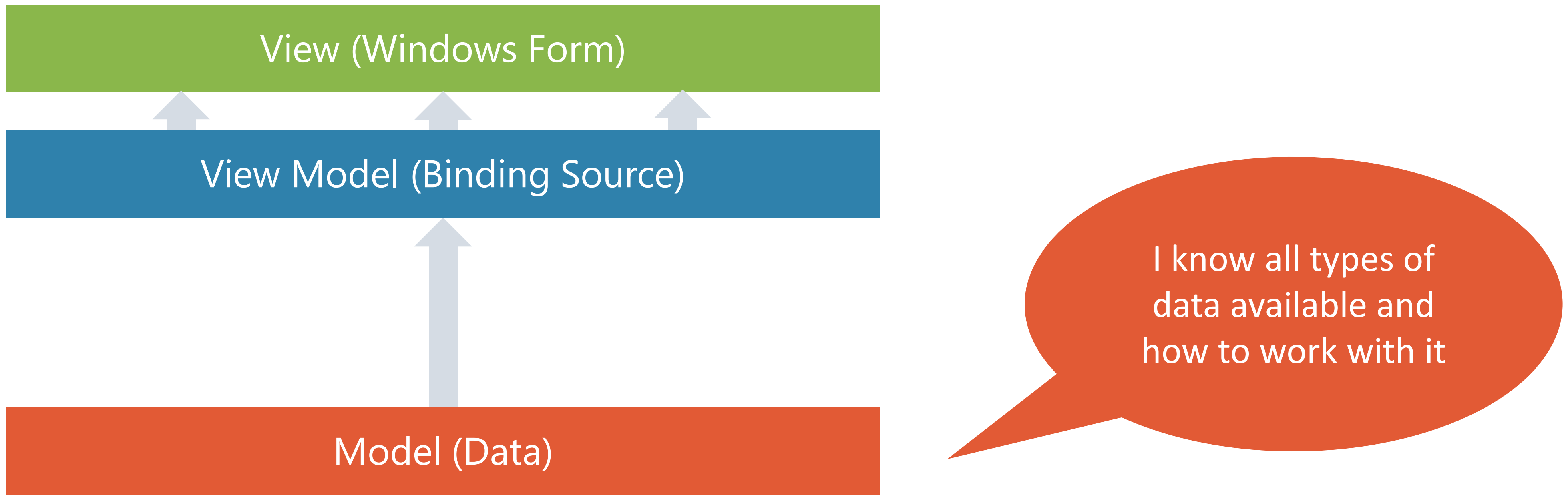
Expression-bodied calculated property using LINQ extension method with a lambda expression as the parameter

MVVM – Model/View/ViewModel



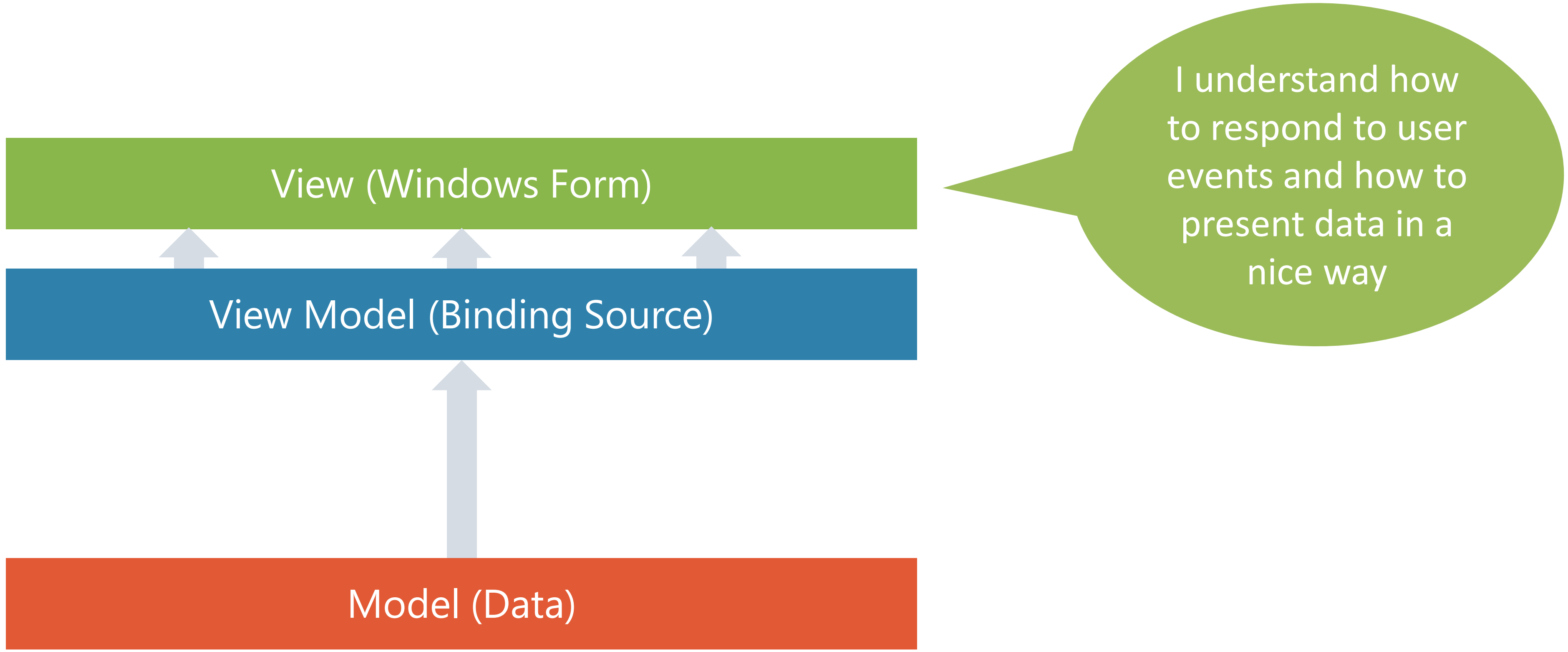
The ViewModel exposes Properties which are bound to UI elements in the Form

MVVM – Model/View/ViewModel

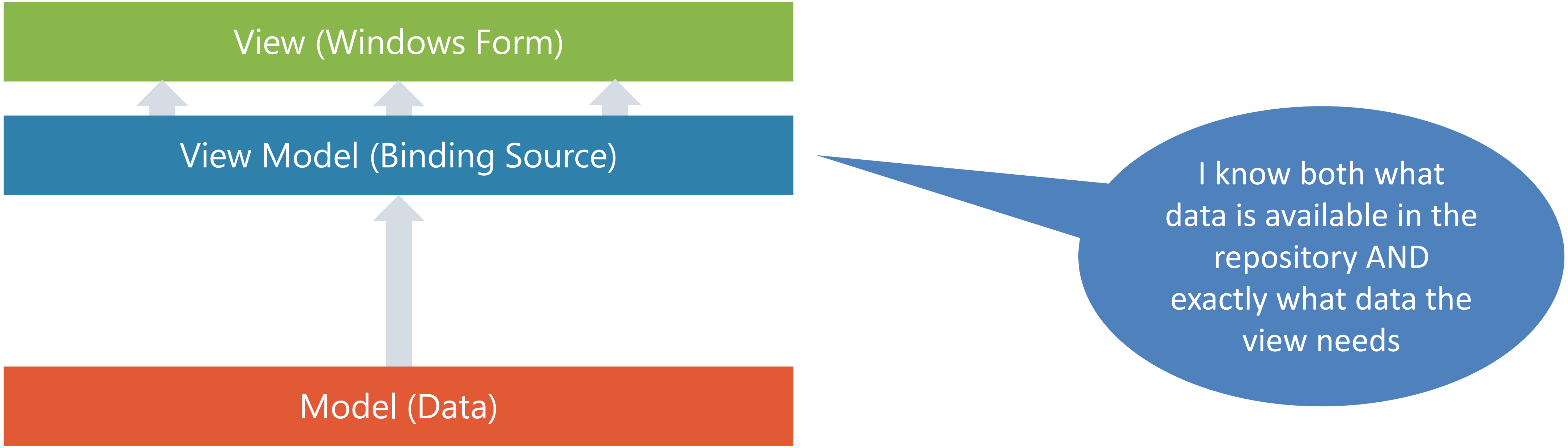


I know all types of data available and how to work with it

MVVM – Model/View/ViewModel

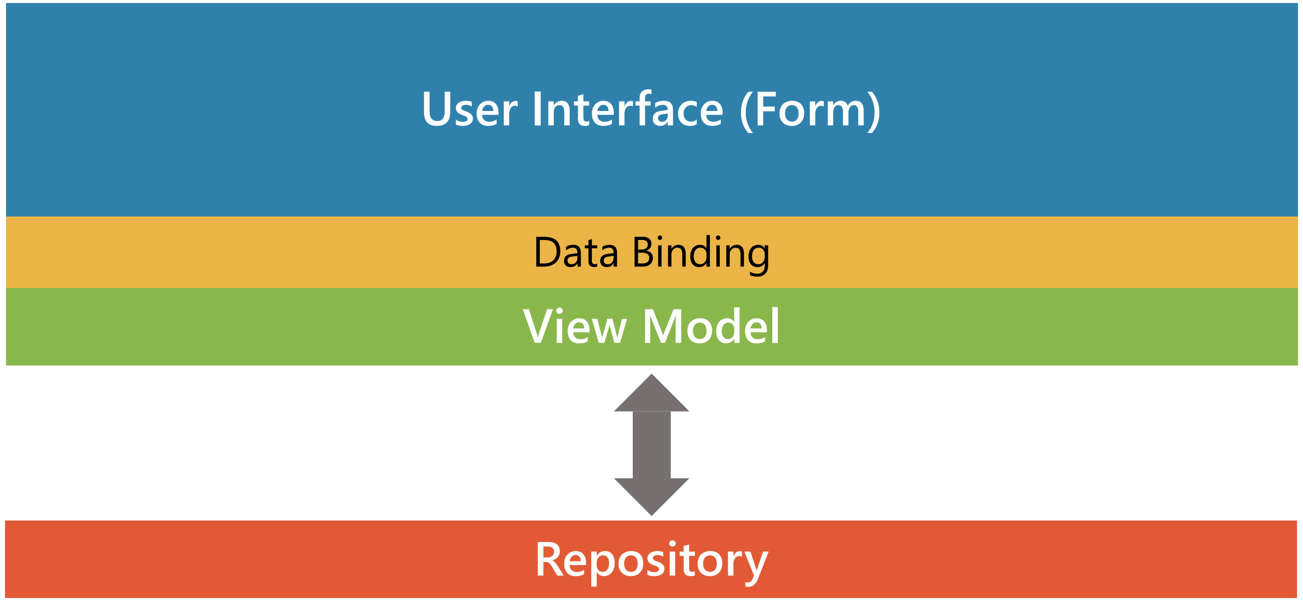

View (Windows Form)
View Model (Binding Source)
Model (Data)
I understand how to respond to user events and how to present data in a nice way

MVVM – Model/View/ViewModel



I know both what data is available in the repository AND exactly what data the view needs

Assignment 7 Architecture (Part A)



I connect the data properties of the controls in the View to the data in the View Model

GUI Design

Great Applications Don’t Happen by Accident

> *“The effort required to use a well-designed application is inversely proportional to the effort required to build one.”*
>
> ***- anonymous***

Developing an Application that is intuitive and easy for a user to operate is usually a lot of work for the developer.

GUI Design

Details to Consider

Form Size

- Fixed?
- Maximizable?
- Minimum Size?

Control Placement

- Alignment
- Spacing
- Anchoring

Keyboard Navigation

- Mnemonics (Accessor Keys)
- Tab Order
- Accept Button
- Cancel Button

Consistency

- With Rest of Application
- With Other Applications