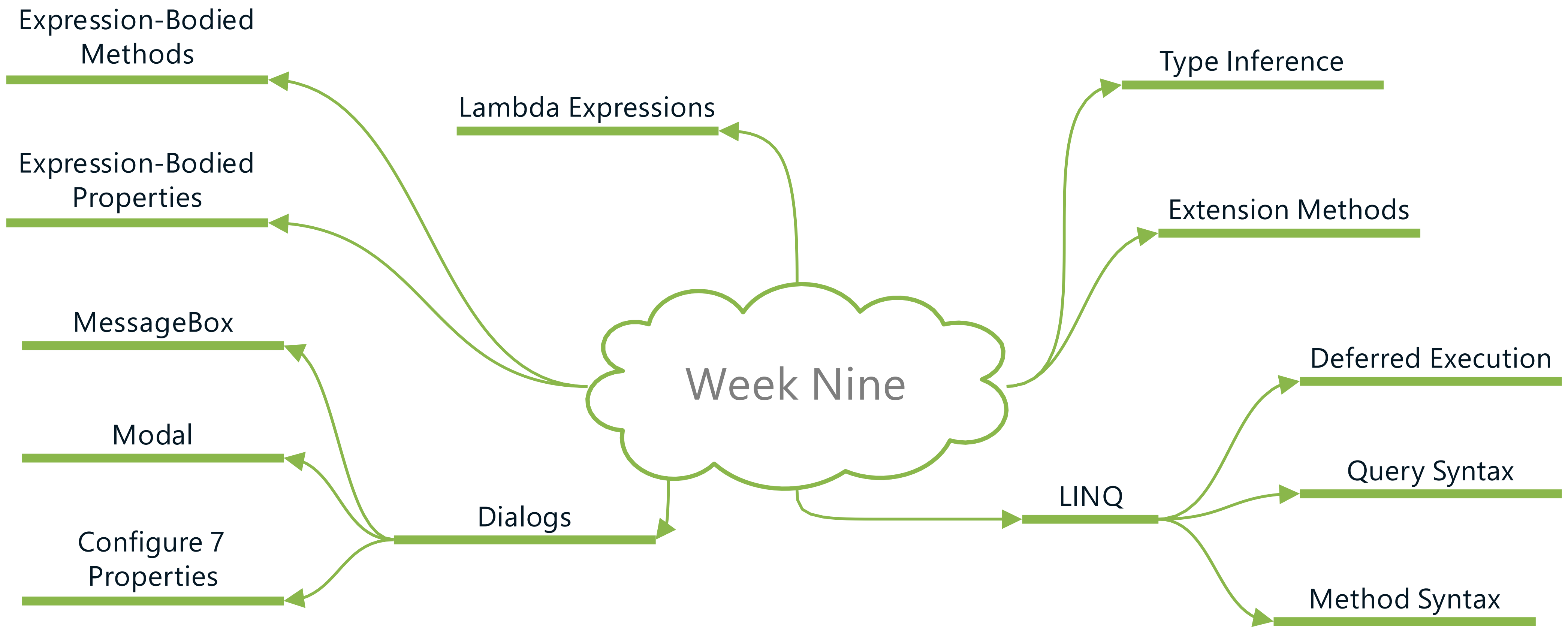
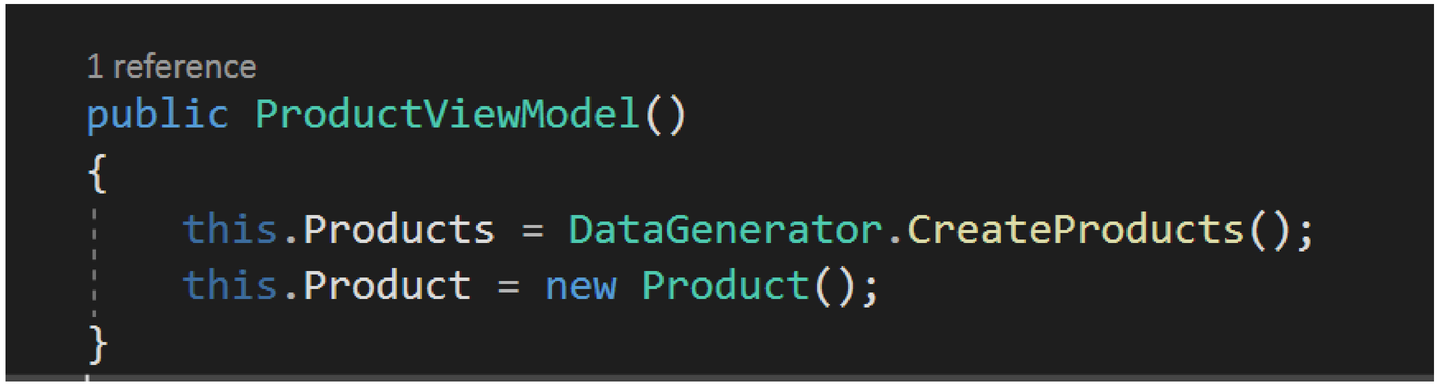
# COMP 3602
C# Application Development

Week Nine

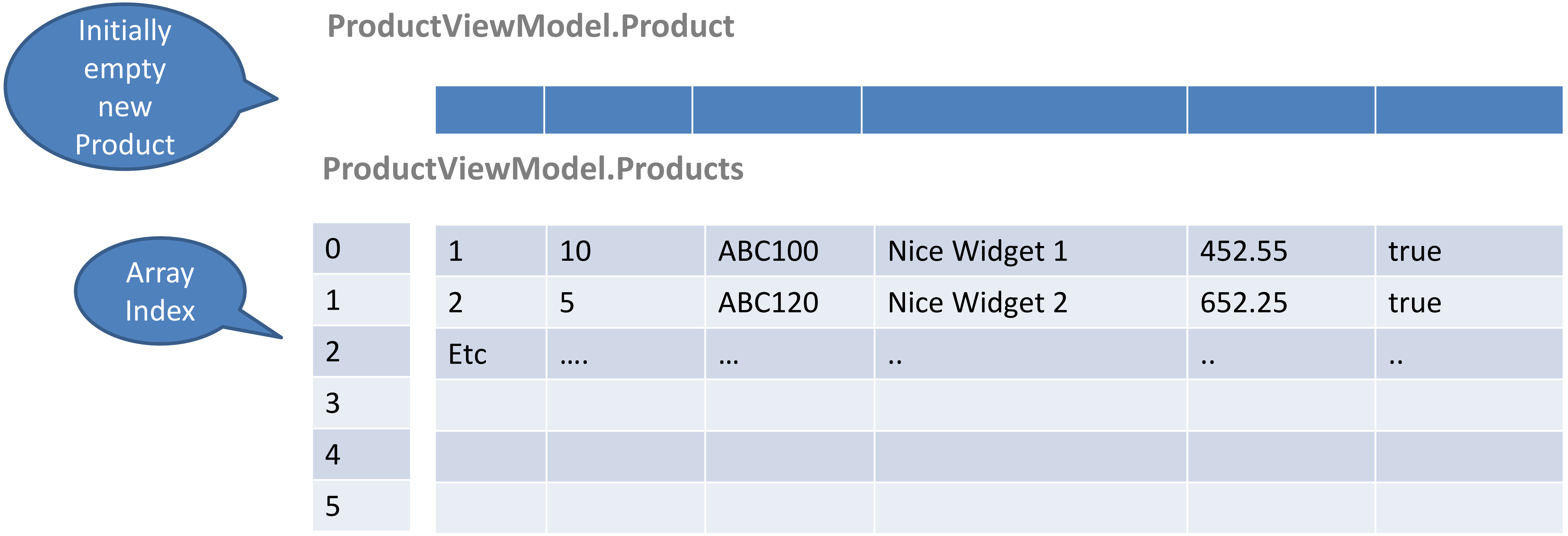
## Tonight's Learning Outcomes



## Product ViewModel explained

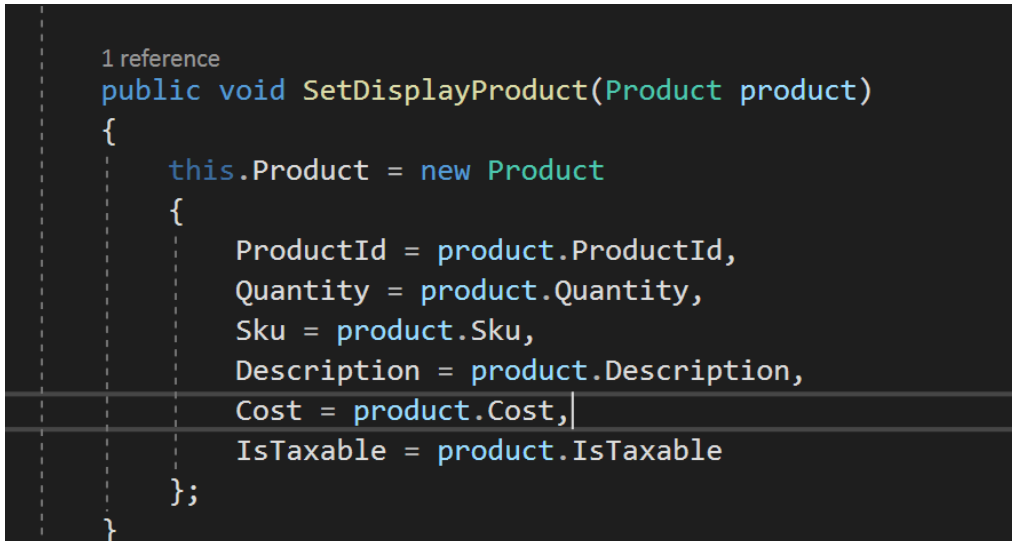


```
1 reference
public ProductViewModel()
{
    this.Products = DataGenerator.CreateProducts();
    this.Product = new Product();
}
```



| | | | | | | |
|---|---|---|---|---|---|---|
| 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



```
1 reference
public void SetDisplayProduct(Product product)
{
    this.Product = new Product
    {
        ProductId = product.ProductId,
        Quantity = product.Quantity,
        Sku = product.Sku,
        Description = product.Description,
        Cost = product.Cost,
        IsTaxable = product.IsTaxable
    };
}
```

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

ProductViewModel.DisplayProduct

| | | | | | |
|---|---|---|---|---|---|
| **2** | **5** | **ABC120** | **Nice Widget 2** | **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;
Product product = productVM.Products[index];
productVM.SetDisplayProduct(product);
```

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

**ProductViewModel.Product**

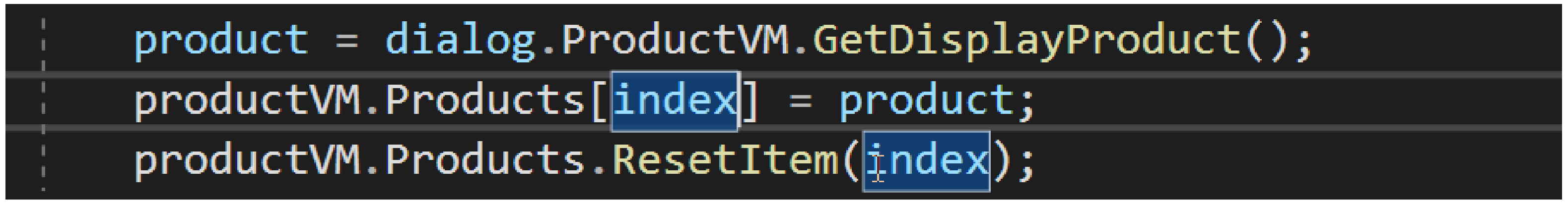
| | | | | | |
|---|---|---|---|---|---|
| **2** | **500** | **ABC120** | **Nice Widget 1000** | **800.00** | **true** |

**ProductViewModel.Products**

This item is not affected

| | | | | | | |
|---|---|---|---|---|---|---|
| 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



```
product = dialog.ProductVM.GetDisplayProduct();
productVM.Products[index] = product;
productVM.Products.ResetItem(index);
```

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

**ProductViewModel.Product**

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

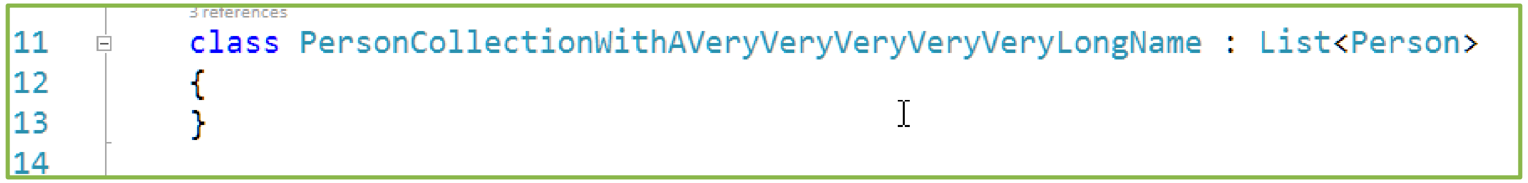
**ProductViewModel.Products**

This item is updated now

| | | | | | | |
|---|---|---|---|---|---|---|
| 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 | | | | | | |

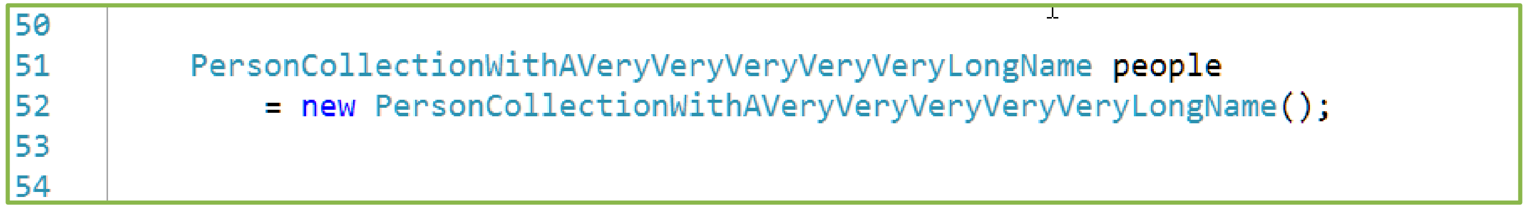
# Type Inference

Collection Class Inherited From List of Type Person



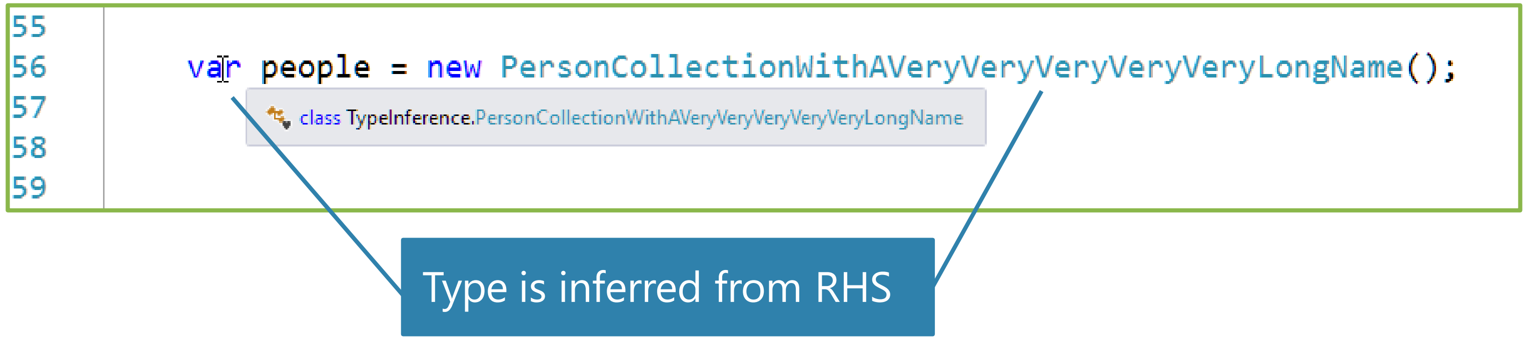
```
3 references
class PersonCollectionWithAVeryVeryVeryVeryVeryLongName : List<Person>
{
}
```

Conventional Assignment Statement



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

Can Be Rewritten as ...



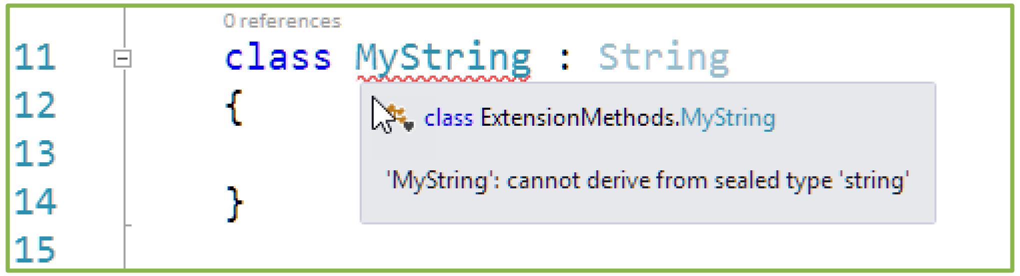
```
var people = new PersonCollectionWithAVeryVeryVeryVeryVeryLongName();
```

class TypeInference.PersonCollectionWithAVeryVeryVeryVeryVeryLongName

- 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

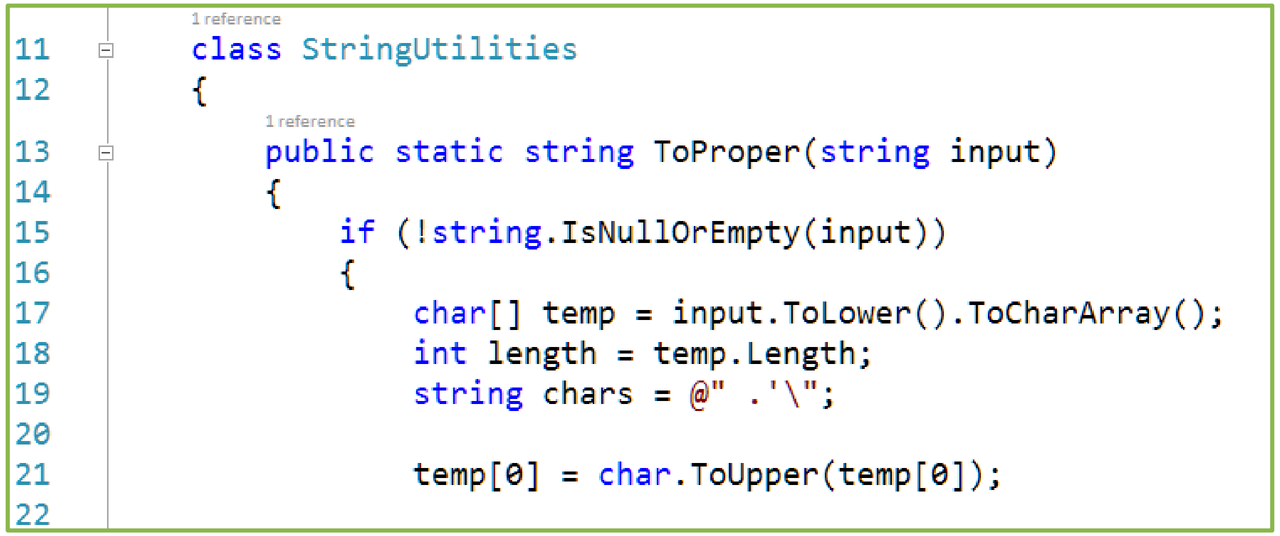
Cannot extend the string class because it is sealed



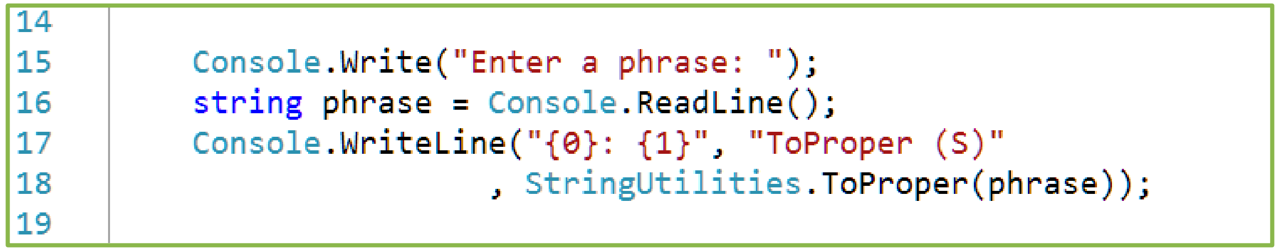
```
0 references
class MyString : String
{

}
```

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]);

```

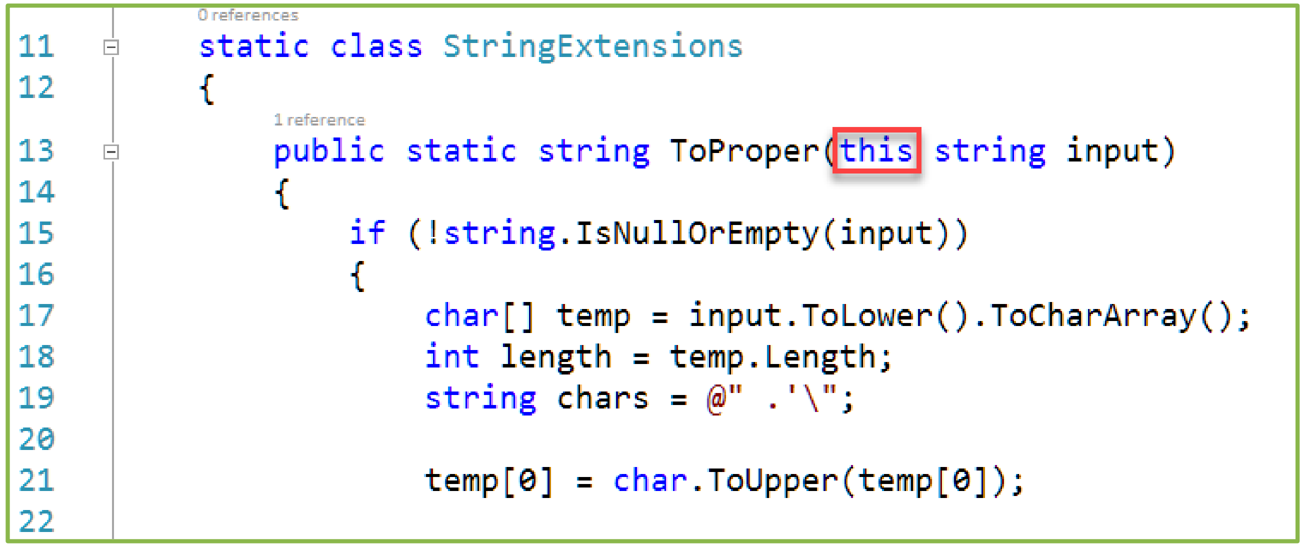


```

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

```

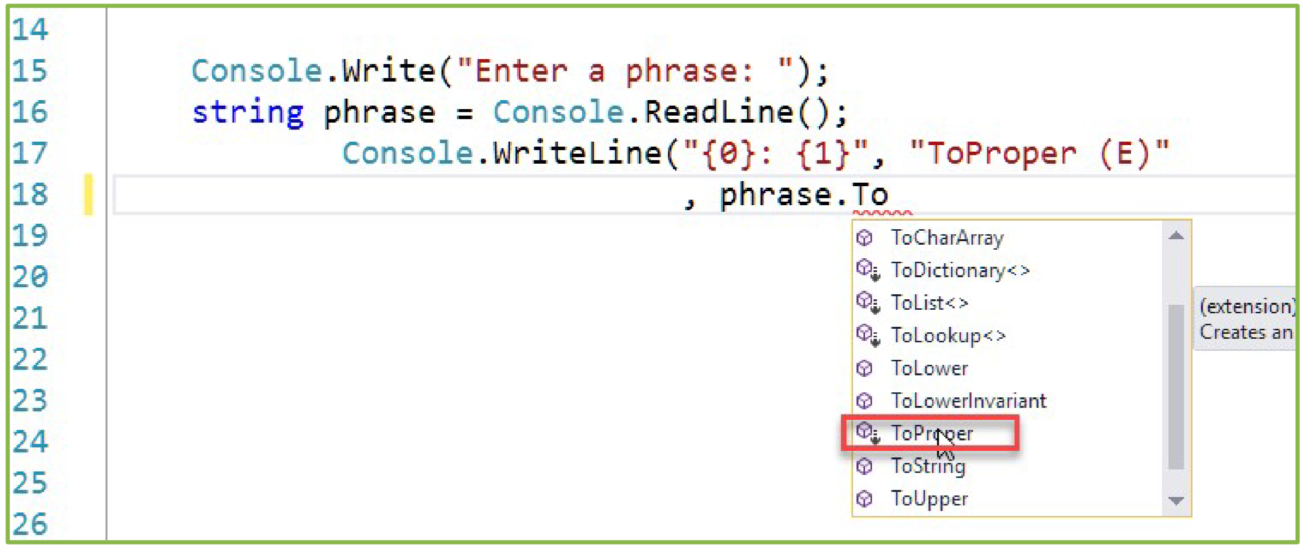
## Static Method

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

# Extension Methods



```
0 references
static class StringExtensions
{
    1 reference
    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

**SQL**

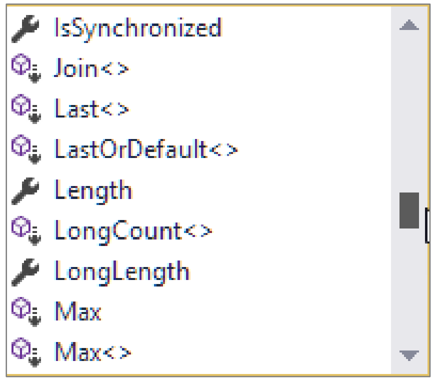
select col1, col2
from table1
where colx = condition

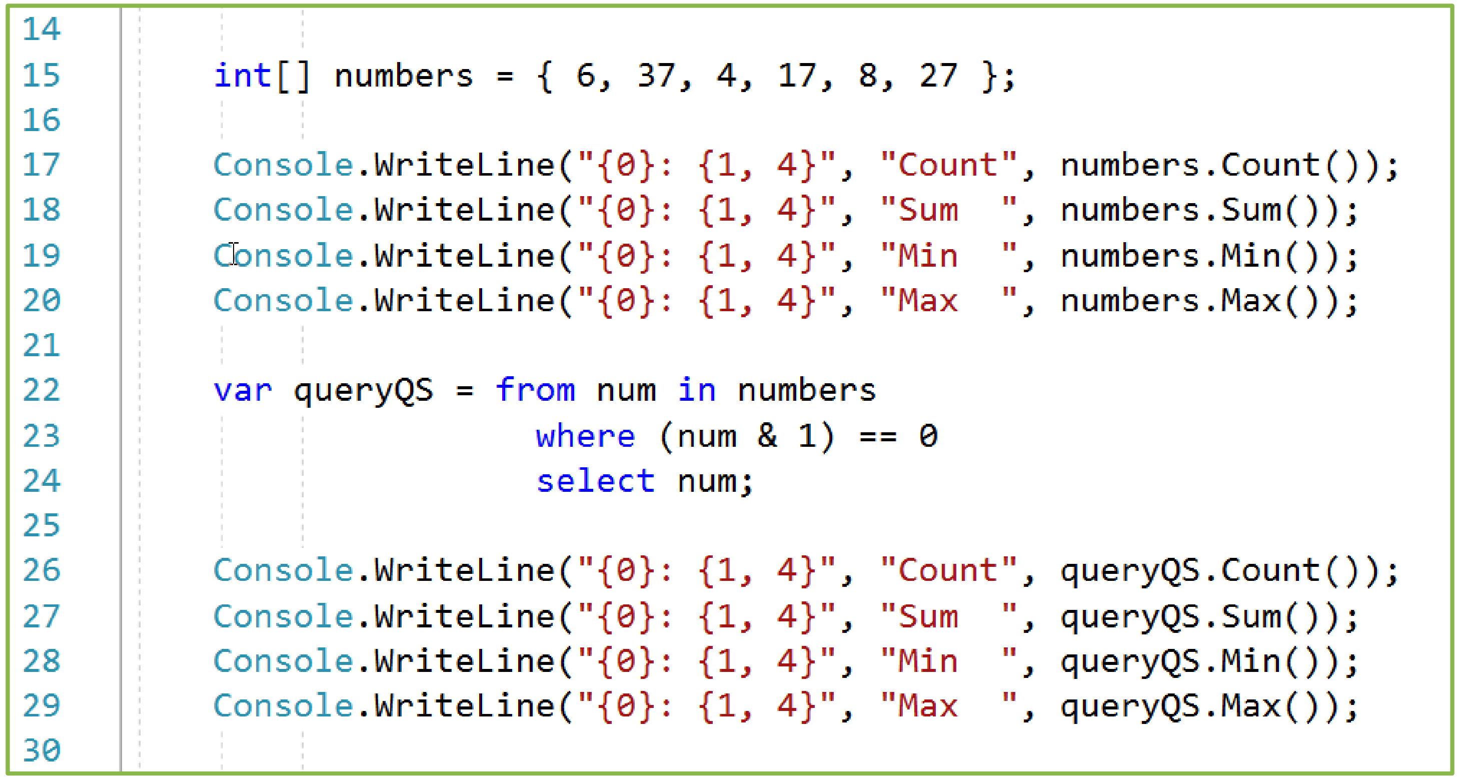
**LINQ**

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

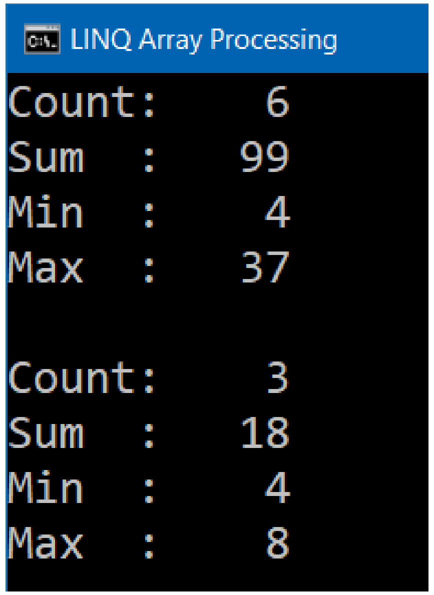
## LINQ - Arrays

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





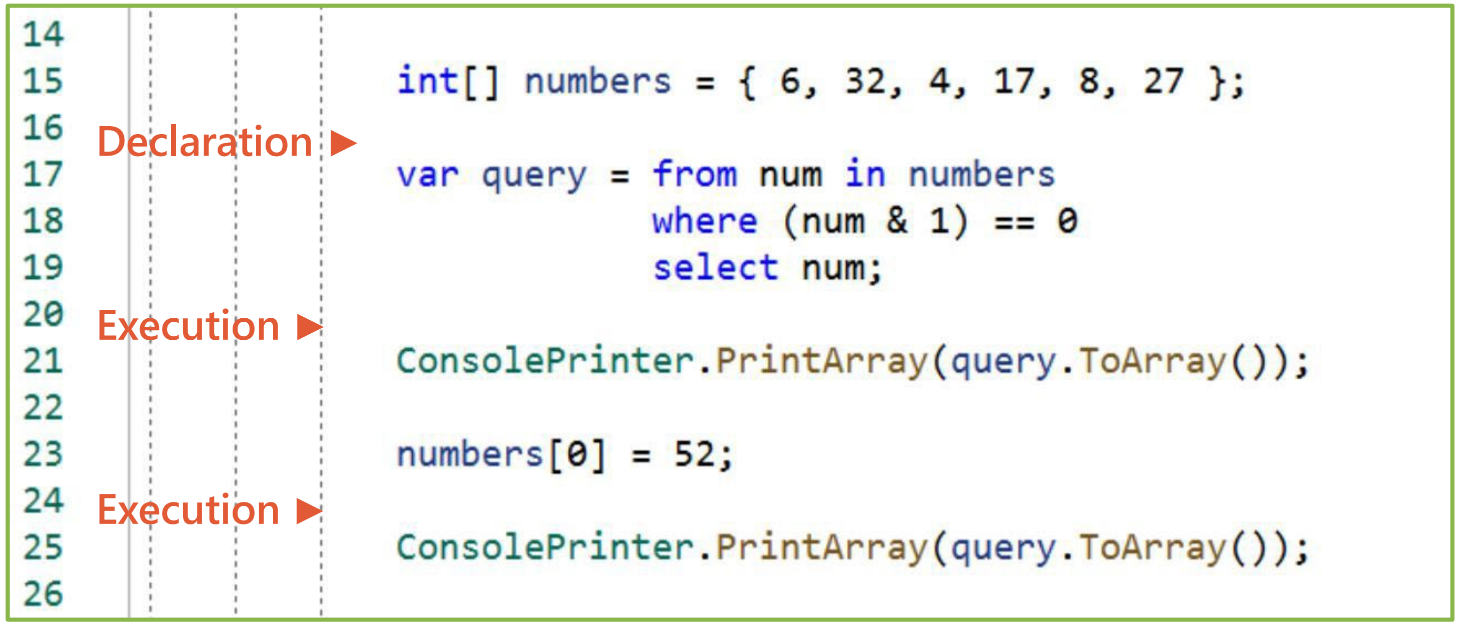
```
    int[] numbers = { 6, 37, 4, 17, 8, 27 };

    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());
```


LINQ Array Processing

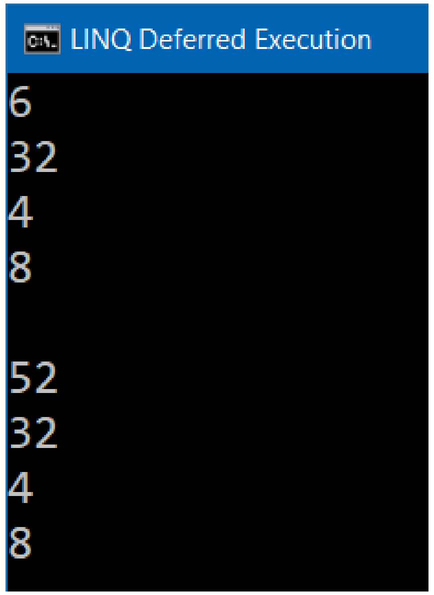
```
Count:    6
Sum  :   99
Min  :    4
Max  :   37

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

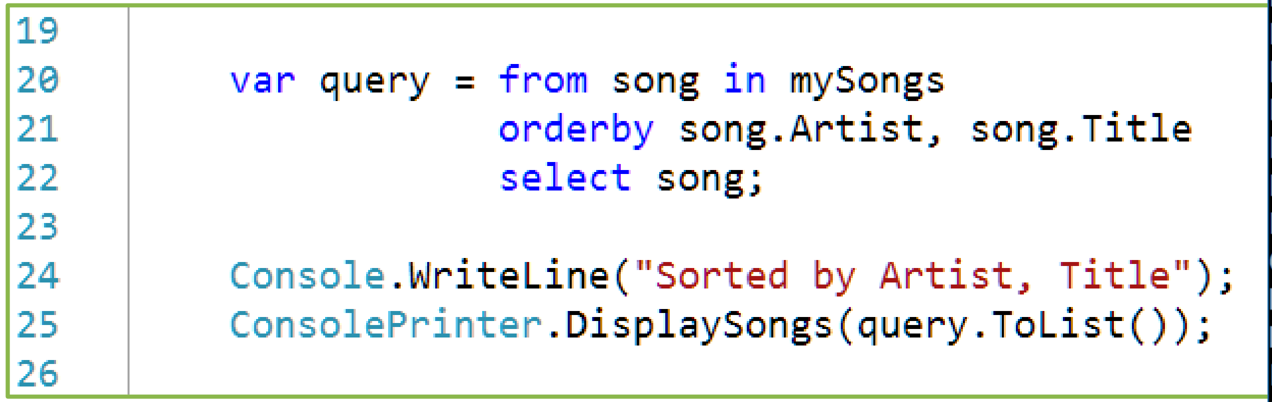


```
int[] numbers = { 6, 32, 4, 17, 8, 27 };

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

ConsolePrinter.PrintArray(query.ToArray());

numbers[0] = 52;

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

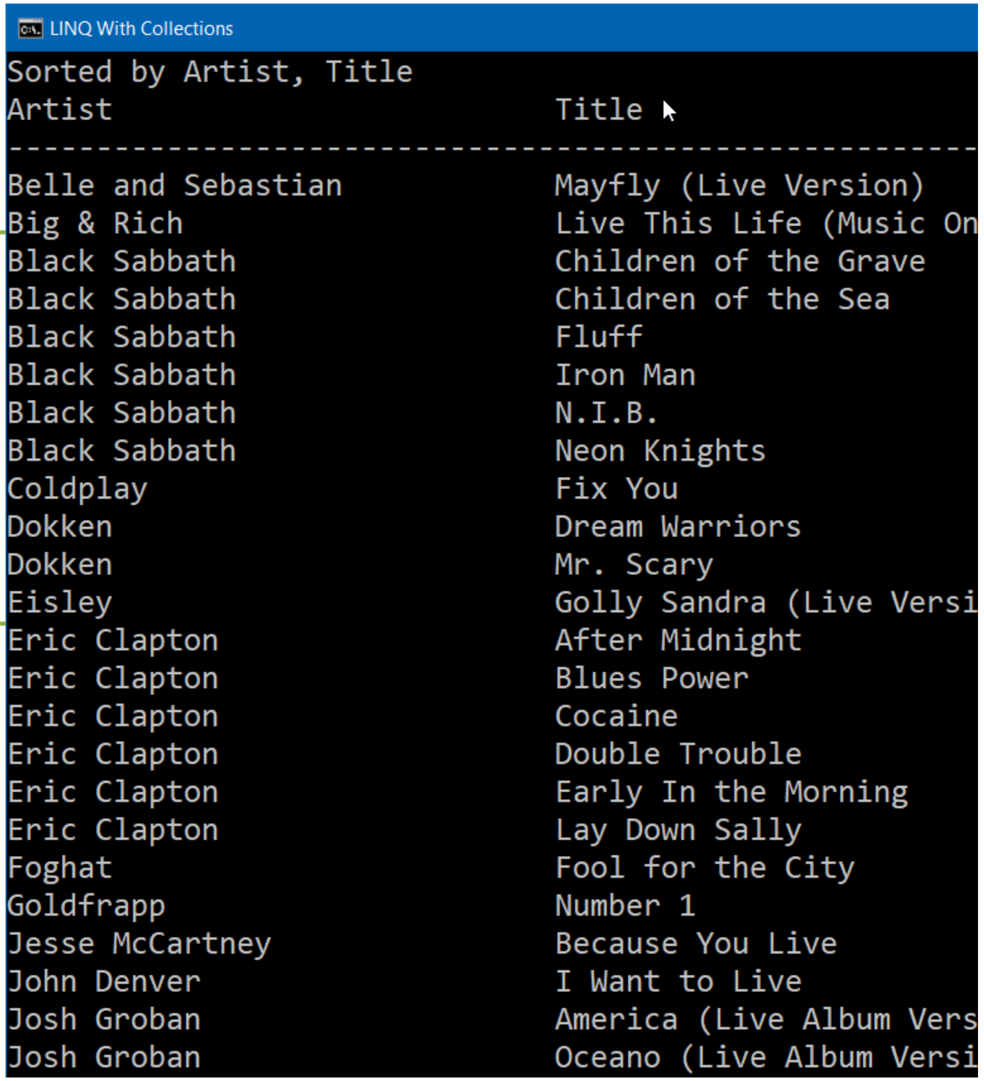


```
6
32
4
8

52
32
4
8
```

## LINQ – Collection Queries



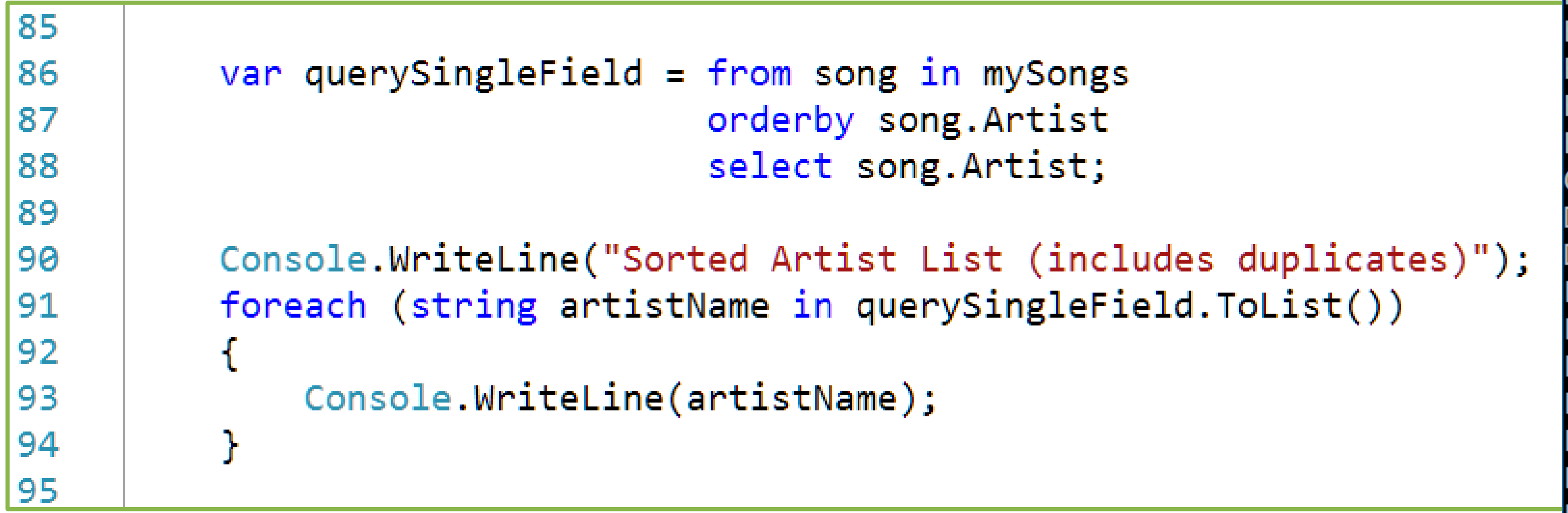
```
    var query = from song in mySongs
                orderby song.Artist, song.Title
                select song;

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



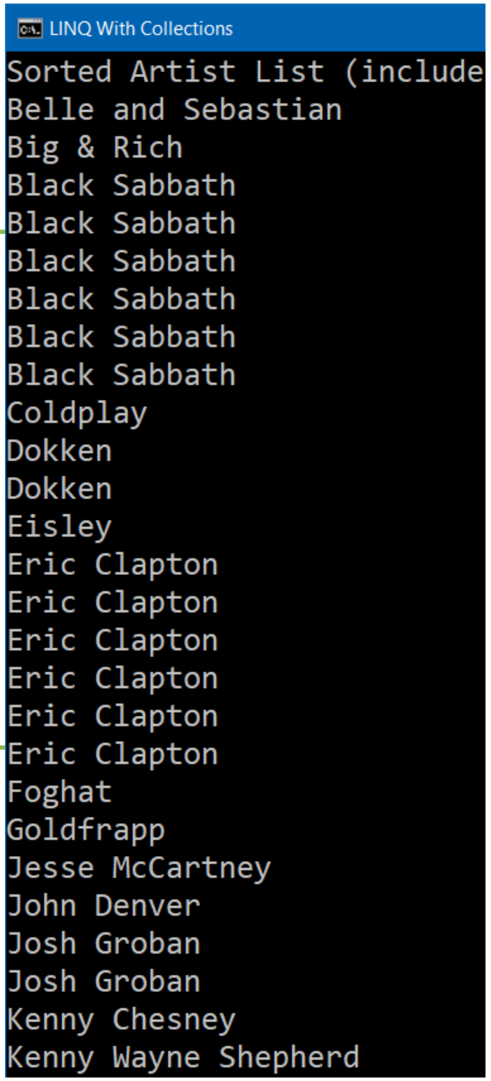
LINQ With Collections

```
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

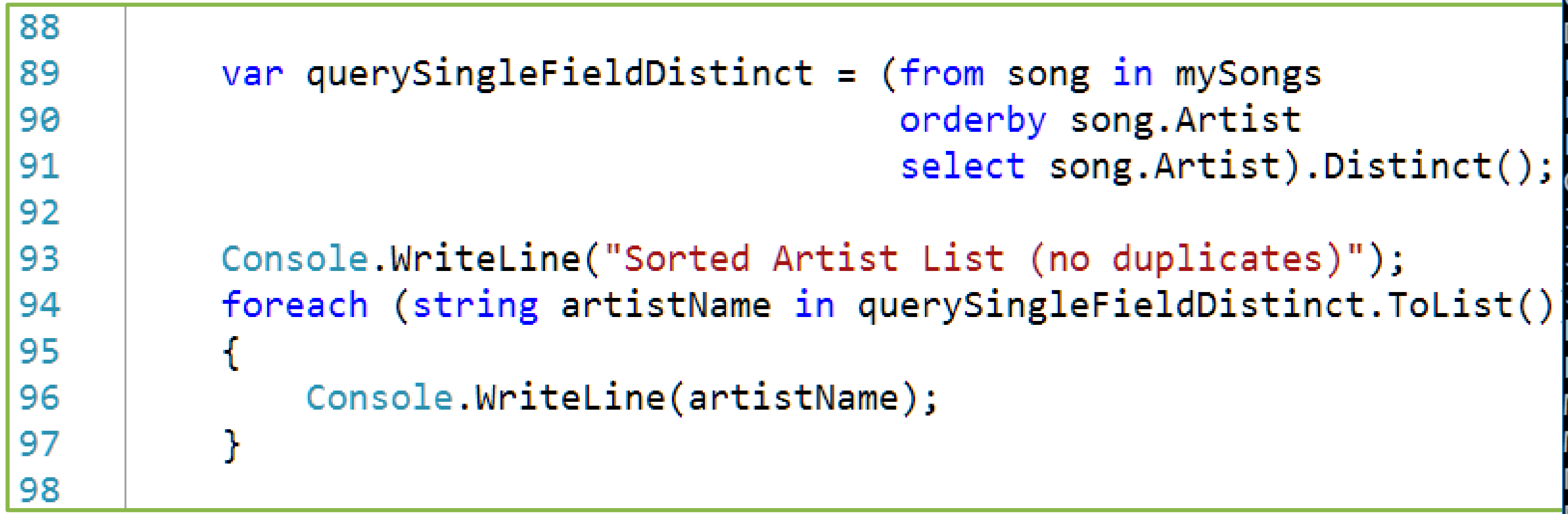


```

    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);
    }

```

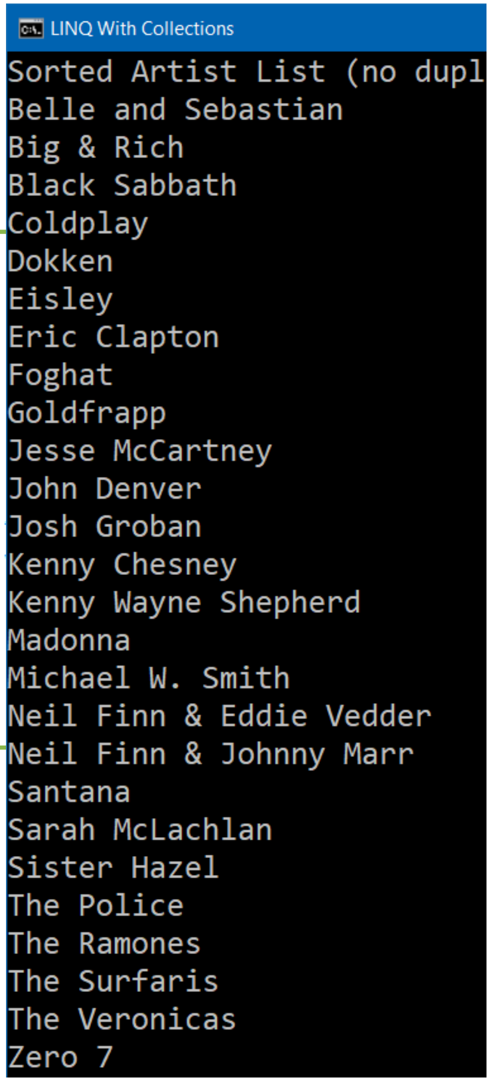


```
Sorted Artist List (include
Belle and Sebastian
Big & Rich
Black Sabbath
Black Sabbath
Black Sabbath
Black Sabbath
Black Sabbath
Black Sabbath
Coldplay
Dokken
Dokken
Eisley
Eric Clapton
Eric Clapton
Eric Clapton
Eric Clapton
Eric Clapton
Eric Clapton
Foghat
Goldfrapp
Jesse McCartney
John Denver
Josh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

## LINQ – Collection Queries



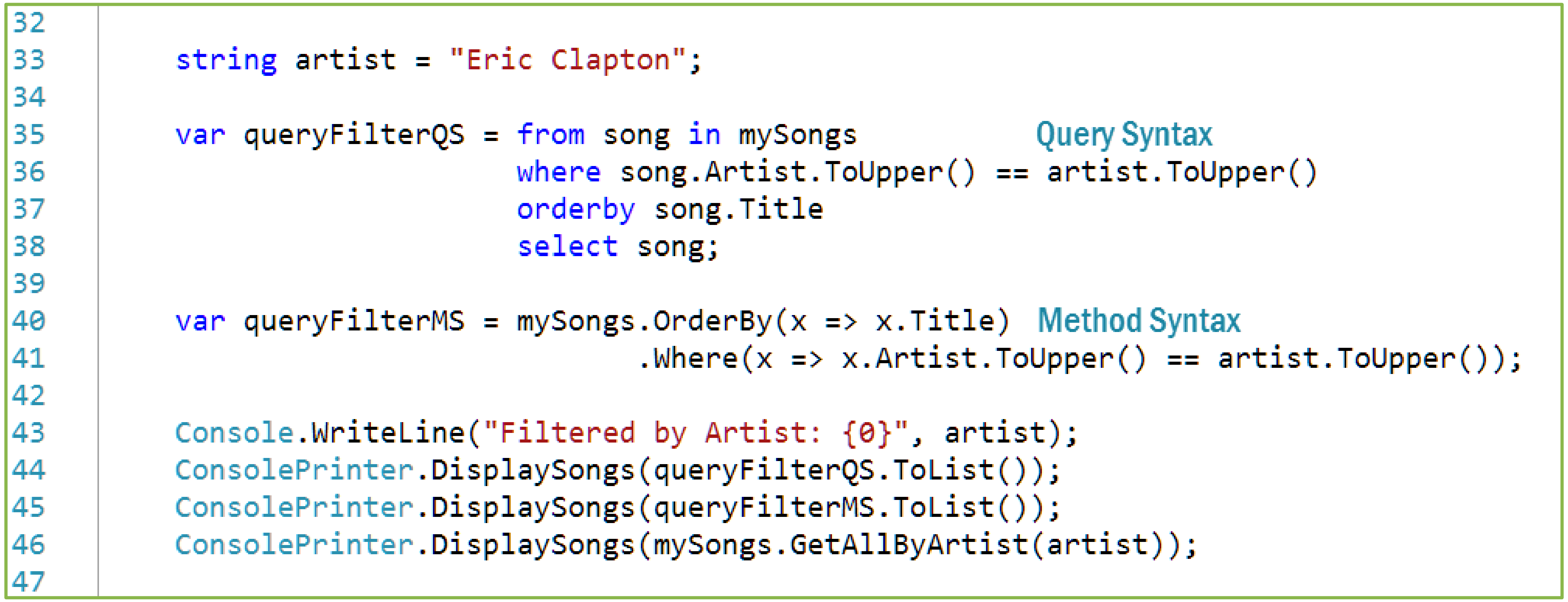
```
var querySingleFieldDistinct = (from song in mySongs
                                orderby song.Artist
                                select song.Artist).Distinct();

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



LINQ With Collections

```
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



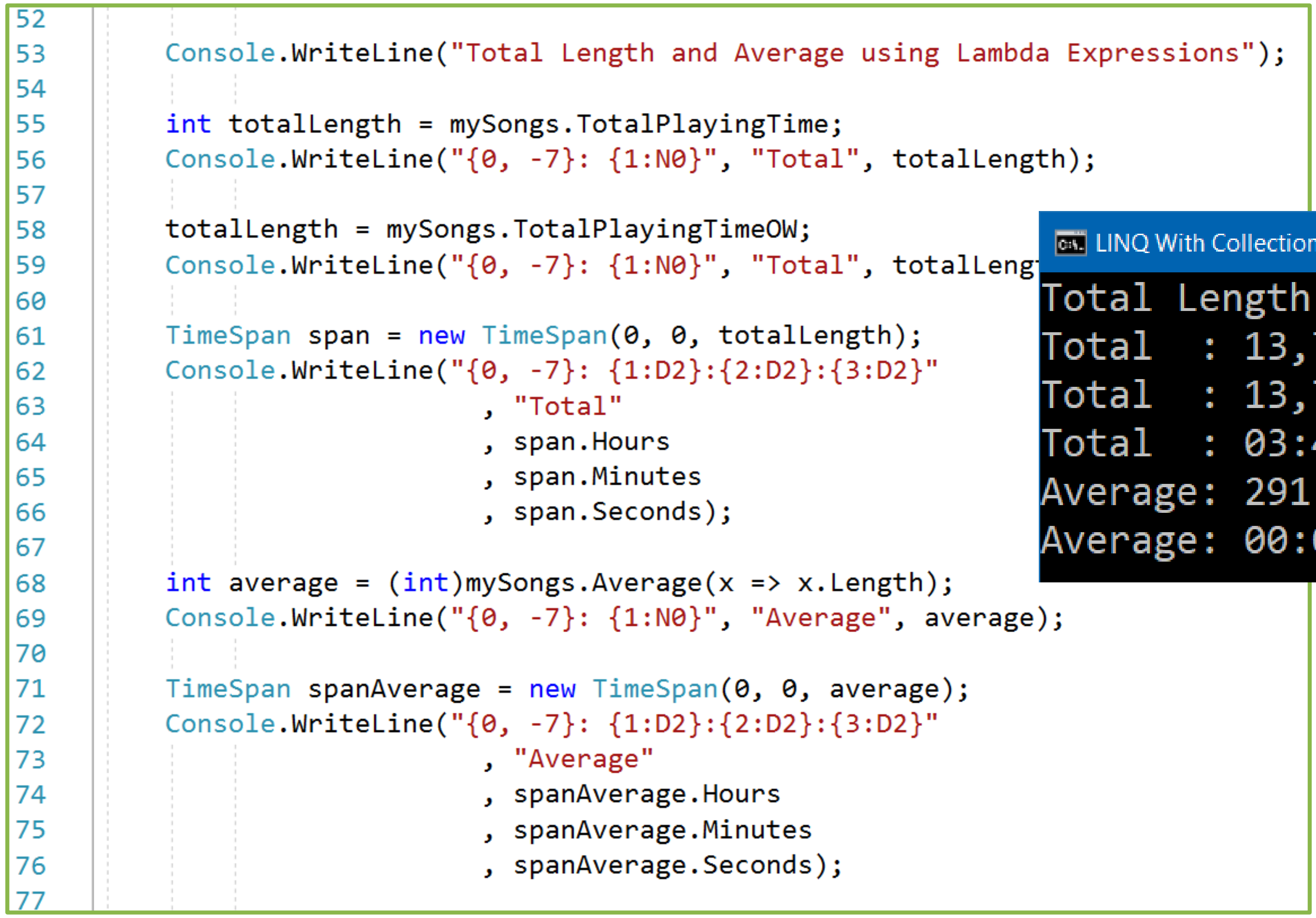
```csharp
string artist = "Eric Clapton";

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

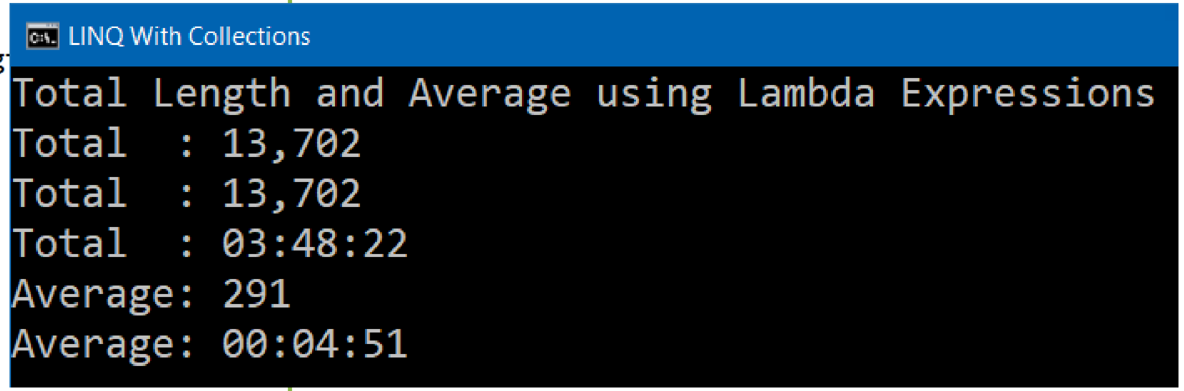
```
  (param => method)
Sum(x => x.Length)
```

x “goes to” x.Length

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



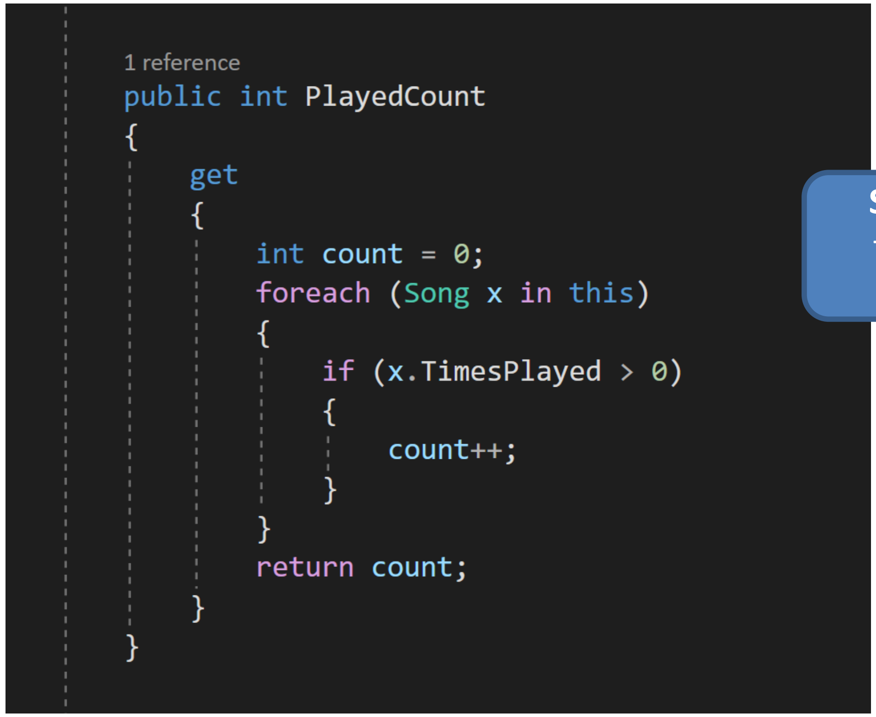
```
    Console.WriteLine("Total Length and Average using Lambda Expressions");

    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
```

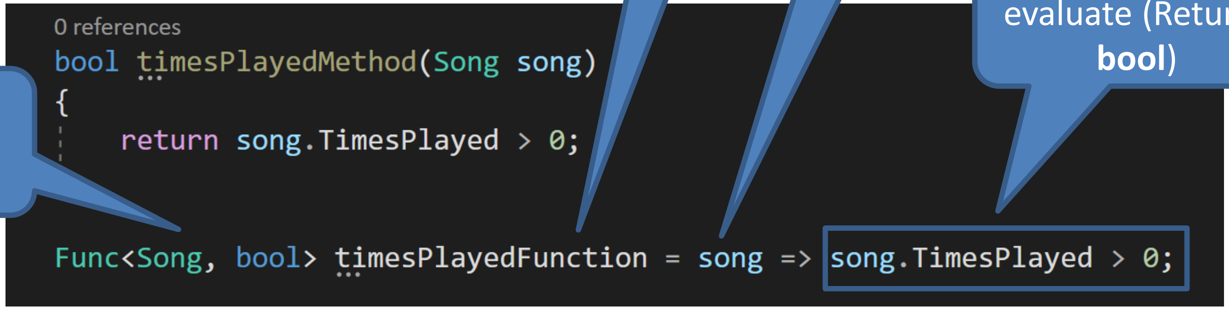
# Lambda Expressions

## Calculated Property



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

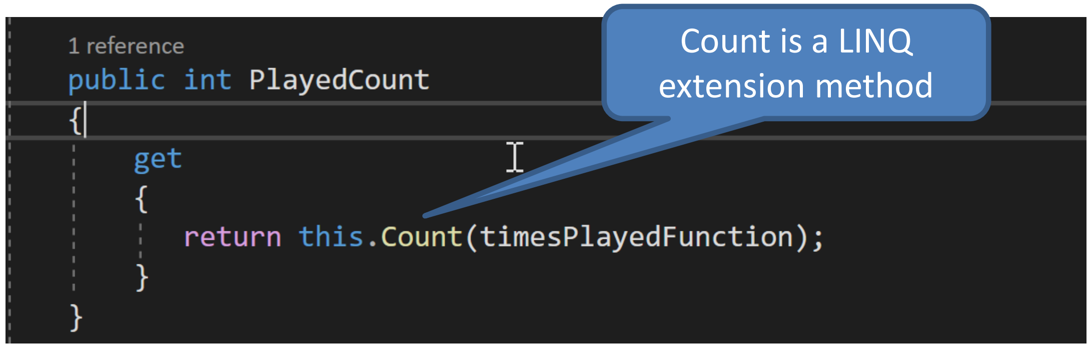
## Method vs Func<t> delegate



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

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

**Song** is input type, **bool** is output type
Reference label
Input parameter (name song, type **Song**)
Method to evaluate (Returns **bool**)

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

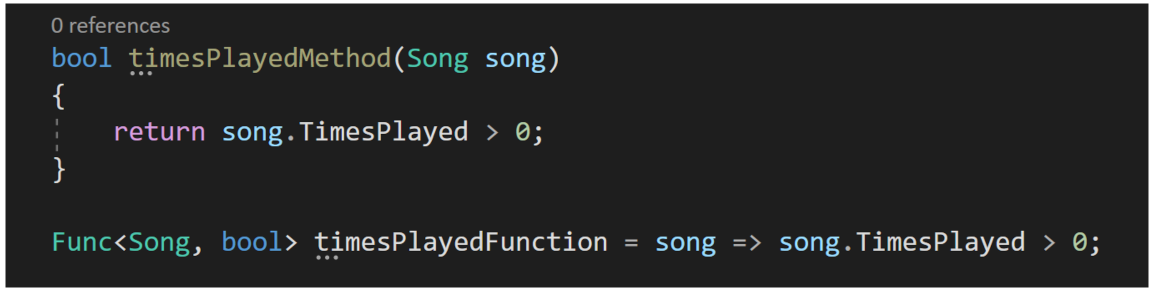


```
1 reference
public int PlayedCount
{
    get
    {
        return this.Count(timesPlayedFunction);
    }
}
```

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

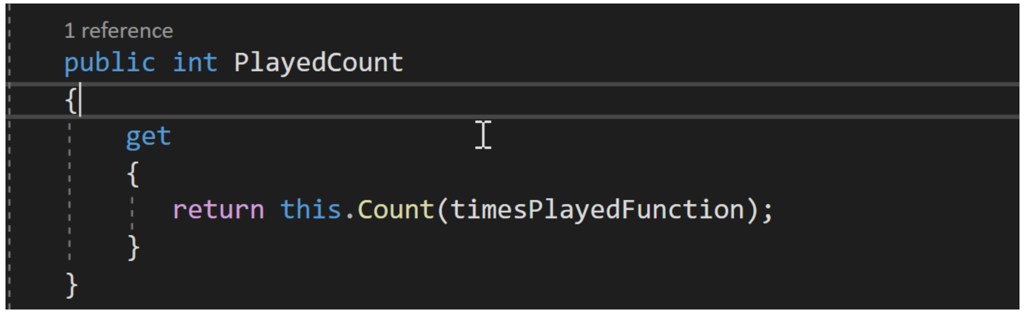
## Lambda Expressions

Method vs Func<t> delegate



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

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

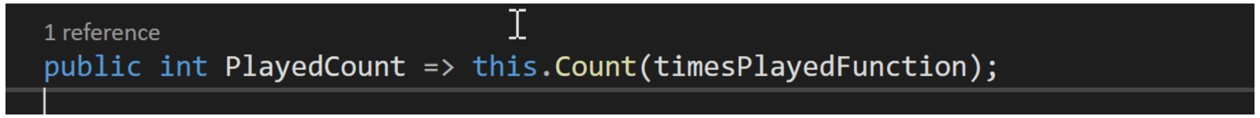
Func<T> can be used to create a reference to a method



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

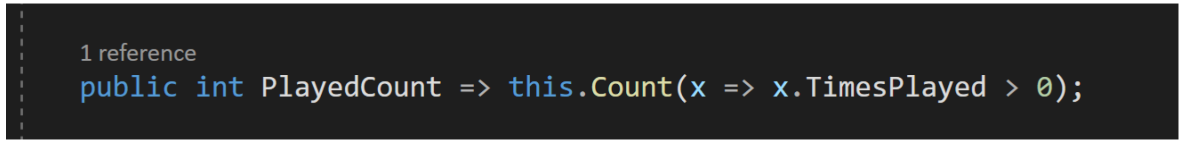
We can then pass this reference in as a parameter to define a method that gets called on each item in a collection

We can use the expression bodied style to save space



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

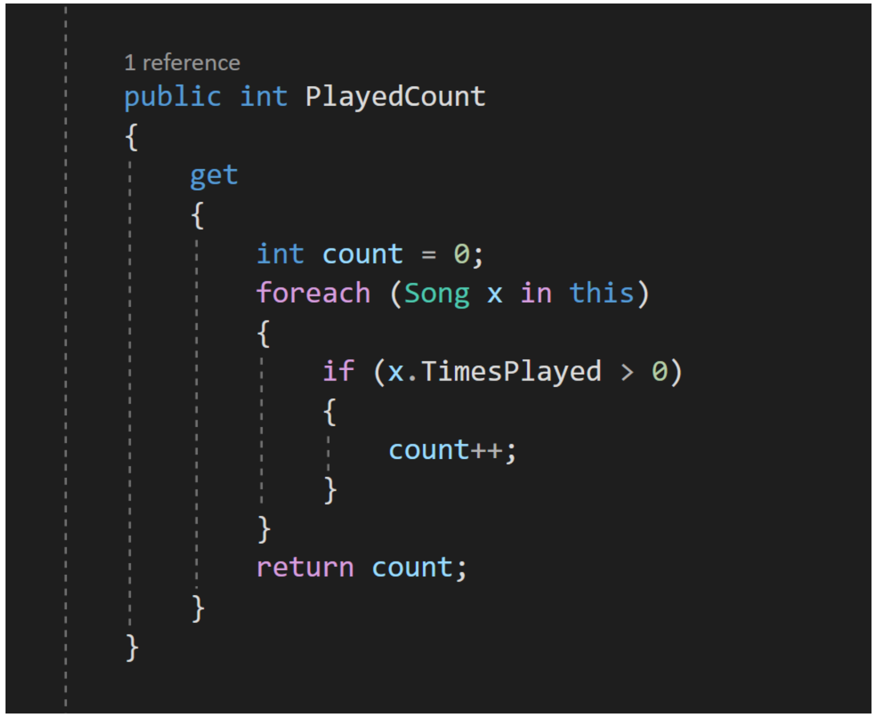
More commonly, instead of defining a Func<T> and referencing it, we just define it inline



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

## Lambda Expressions

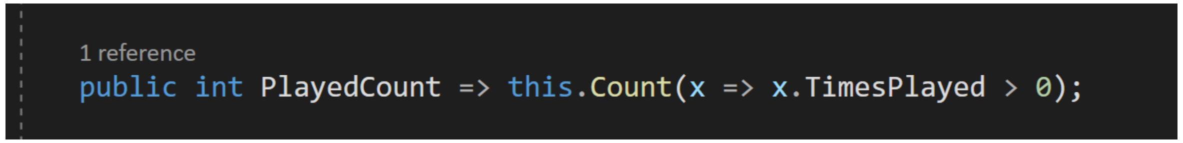
We end up going from this:



```
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:

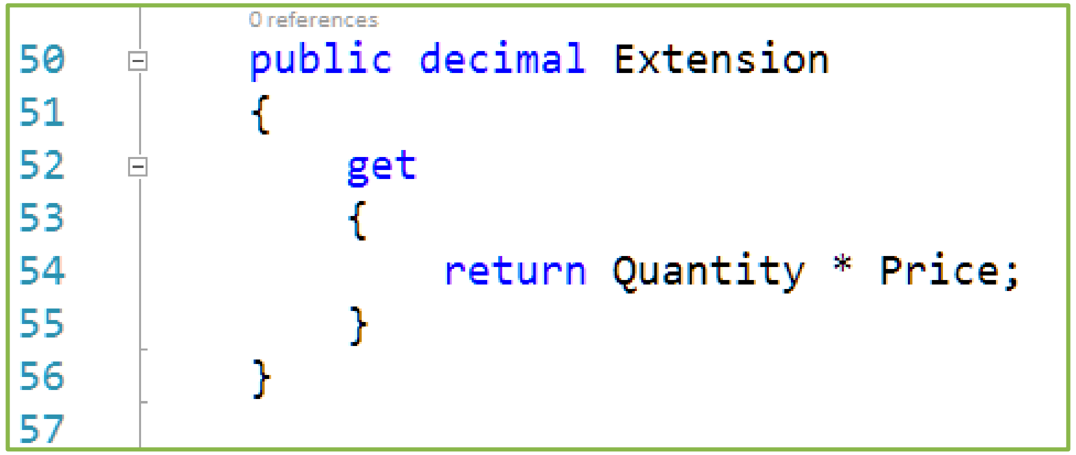


```
1 reference
public int PlayedCount => this.Count(x => x.TimesPlayed > 0);
```

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

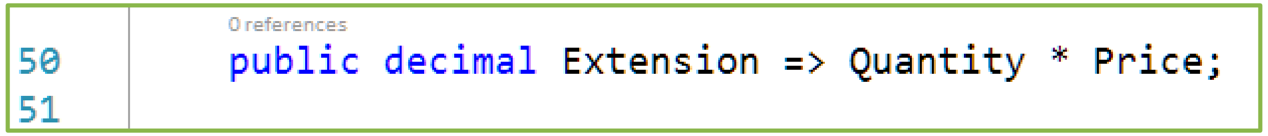
## Expression-Bodied Properties (Calculated)

One-line calculated property …



```
        0 references
        public decimal Extension
        {
            get
            {
                return Quantity * Price;
            }
        }

```

… can be rewritten as



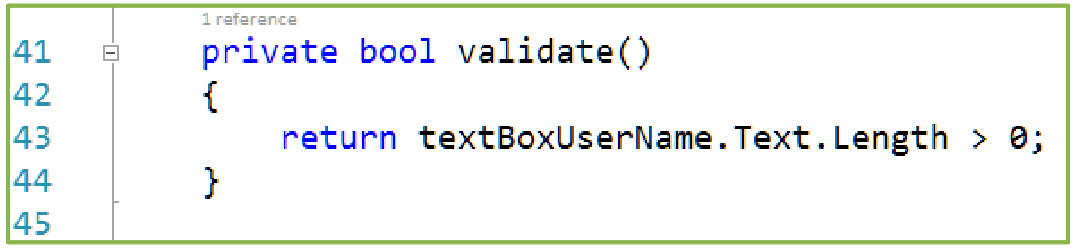
```
        0 references
        public decimal Extension => Quantity * Price;

```

*A one-line method should be written in one line of code.*

*Anders Hejlsberg*
*Chief C# Architect*

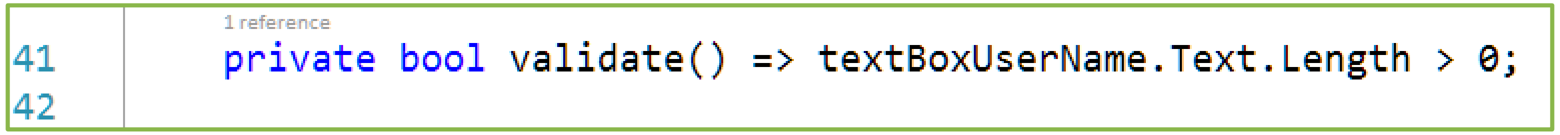
## Expression-Bodied Methods

One-line method …



```
1 reference
private bool validate()
{
    return textBoxUserName.Text.Length > 0;
}
```

… can be rewritten as

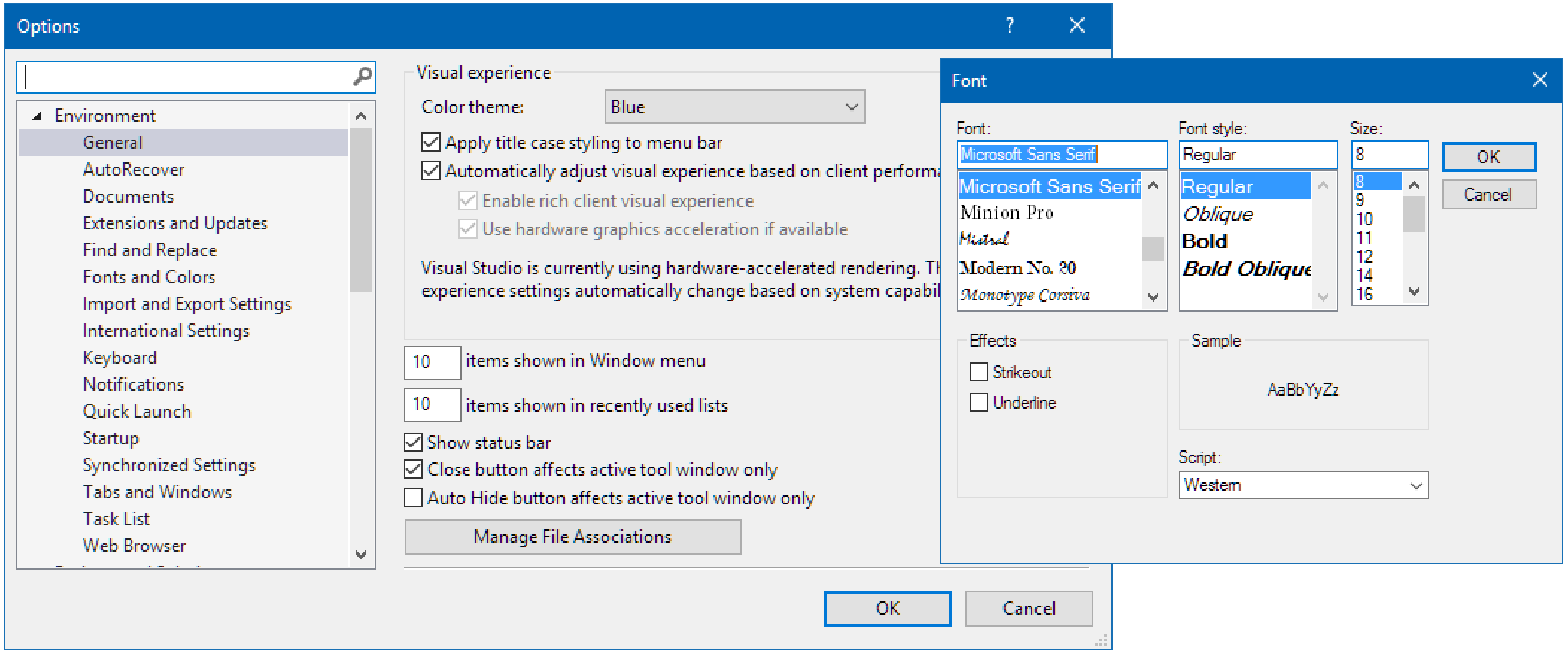


```
1 reference
private bool validate() => textBoxUserName.Text.Length > 0;
```

*A one-line method should be written in one line of code.*

*Anders Hejlsberg*
*Chief C# Architect*

# Dialogs



Visual experience

Color theme: Blue

☑ Apply title case styling to menu bar

☑ Automatically adjust visual experience based on client performa

☑ Enable rich client visual experience

☑ Use hardware graphics acceleration if available

Visual Studio is currently using hardware-accelerated rendering. Th
experience settings automatically change based on system capabil

10 items shown in Window menu

10 items shown in recently used lists

☑ Show status bar

☑ Close button affects active tool window only

☐ Auto Hide button affects active tool window only

Font: Microsoft Sans Serif

Microsoft Sans Serif
Minion Pro
Mistral
Modern No. 20
Monotype Corsiva

Font style: Regular

Regular
Oblique
Bold
Bold Oblique

Size: 8

8
9
10
11
12
14
16

Effects

☐ Strikeout

☐ Underline

Sample

AaBbYyZz

Script: Western

## Dialogs – MessageBox Class

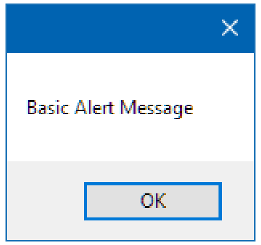
A dialog, or dialog box, is a Form other than the main application form, whose purpose is to display information to the user or to get a response from the user.

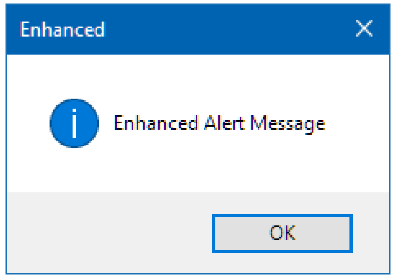
They are called dialogs because they form a dialog between the user and your application.

```
        MessageBox.Show("Basic Alert Message");
```

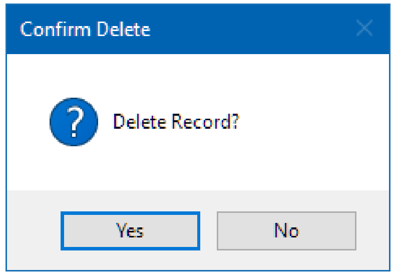
```
        MessageBox.Show("Enhanced Alert Message"
                      , "Enhanced"
                      , MessageBoxButtons.OK
                      , MessageBoxIcon.Information);
```

```
        result = MessageBox.Show("Delete Record?"
                               , "Confirm Delete"
                               , MessageBoxButtons.YesNo
                               , MessageBoxIcon.Question);
```


Basic Alert Message

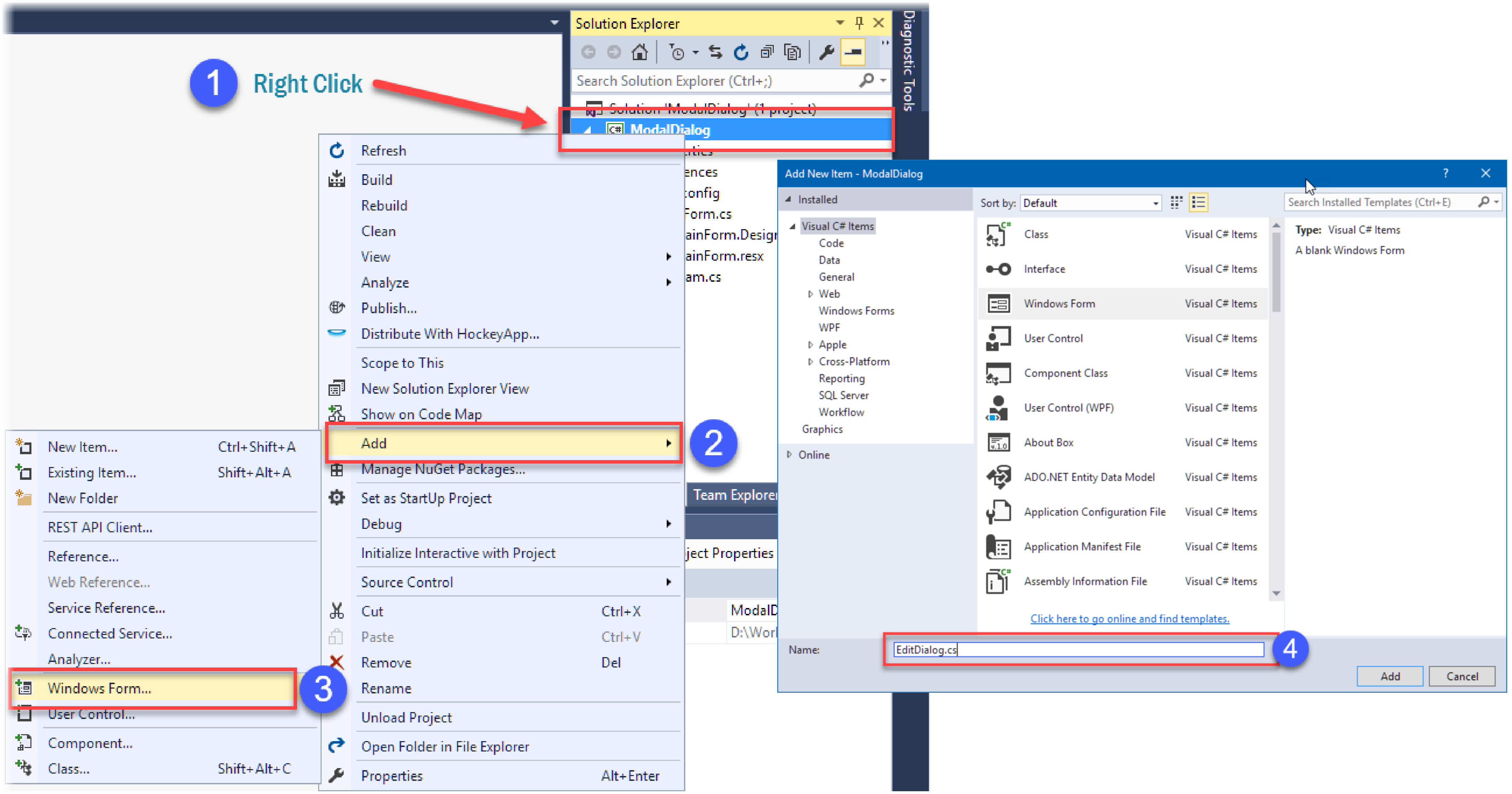

Enhanced

Enhanced Alert Message

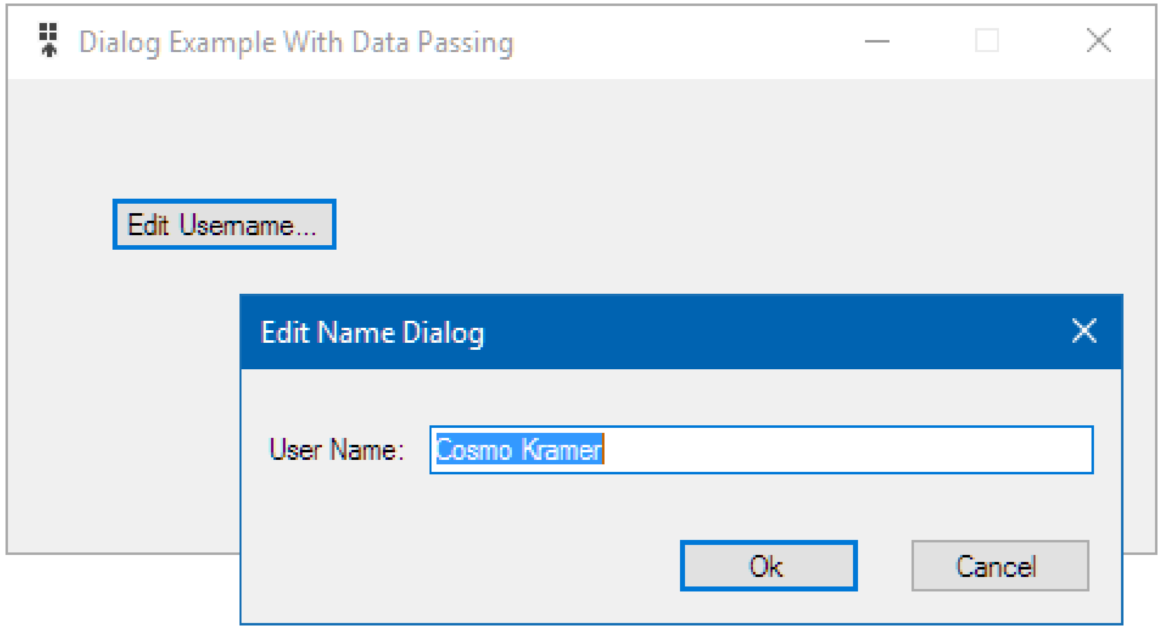

Confirm Delete

Delete Record?

# Adding a New Form



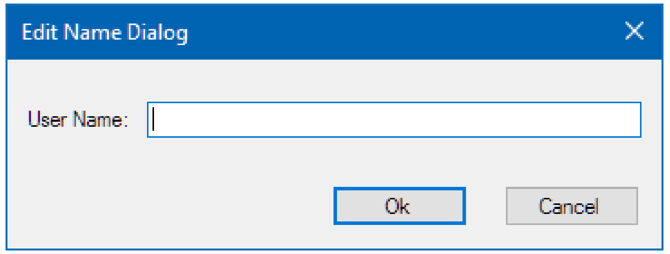
## Modal Dialogs



Modal dialogs force the user to interact with them.  The parent form is non-responsive until the dialog closes.

Call the ShowDialog() method to show a form modally.

## Modal Dialogs – Default Behavior



**Property Checklist**

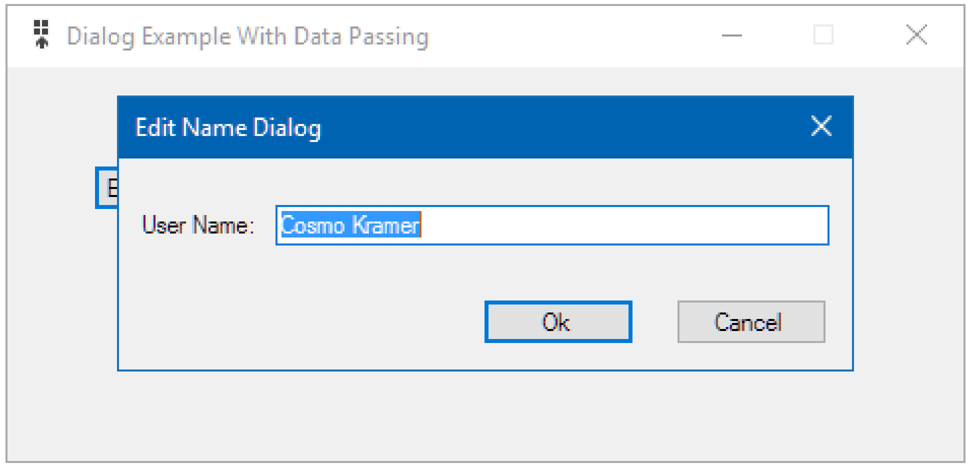
**Modal dialogs have a particular look and feel:**

- Cannot be resized, minimized or maximized
- Positioned center screen or center parent
- OK button (available from Enter key)
- Cancel button (available from Esc key)
- No control box
- No taskbar button

**Set the following seven properties to obtain this behavior:**

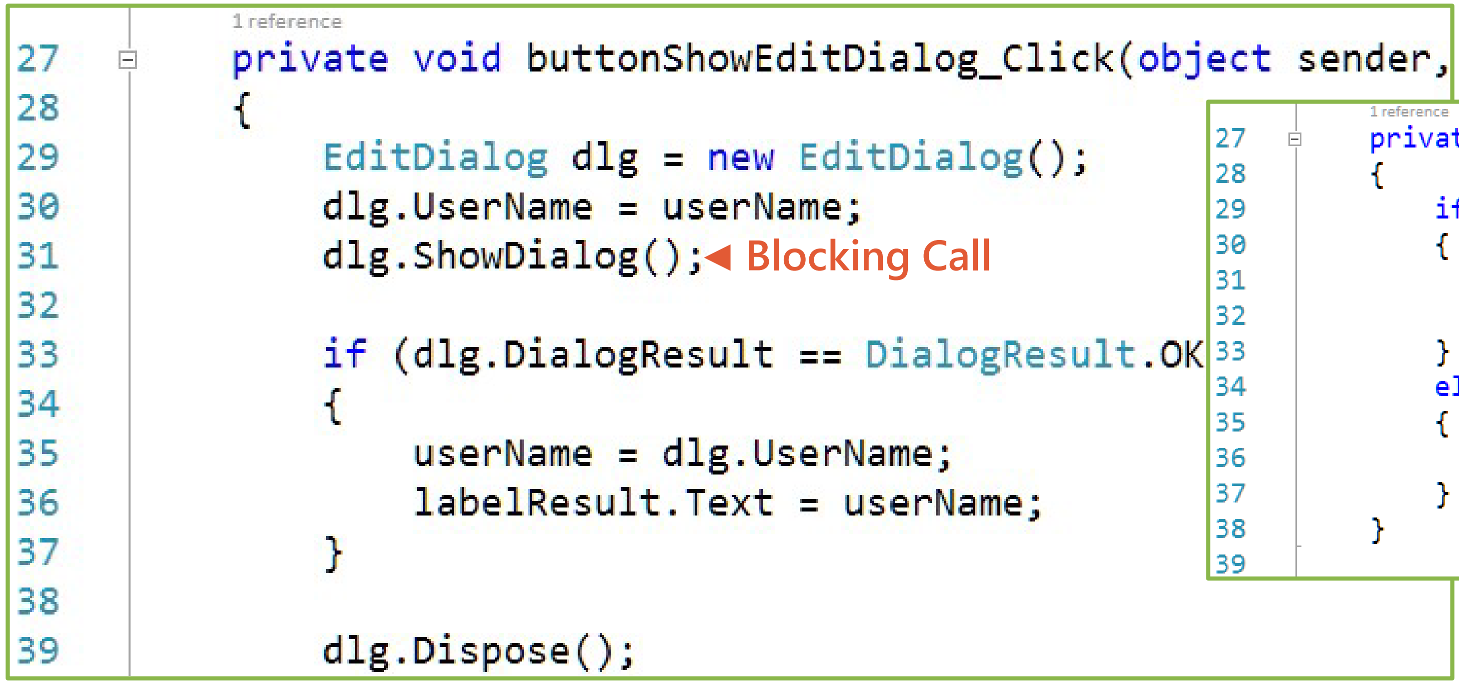
- FormBorderStyle = FixedDialog
- MaximizeBox = false
- MinimizeBox = false
- AcceptButton = buttonOK
- CancelButton = buttonCancel
- StartPosition = CenterScreen or CenterParent
- ShowInTaskbar = false

## Modal Dialogs – Opening and Closing



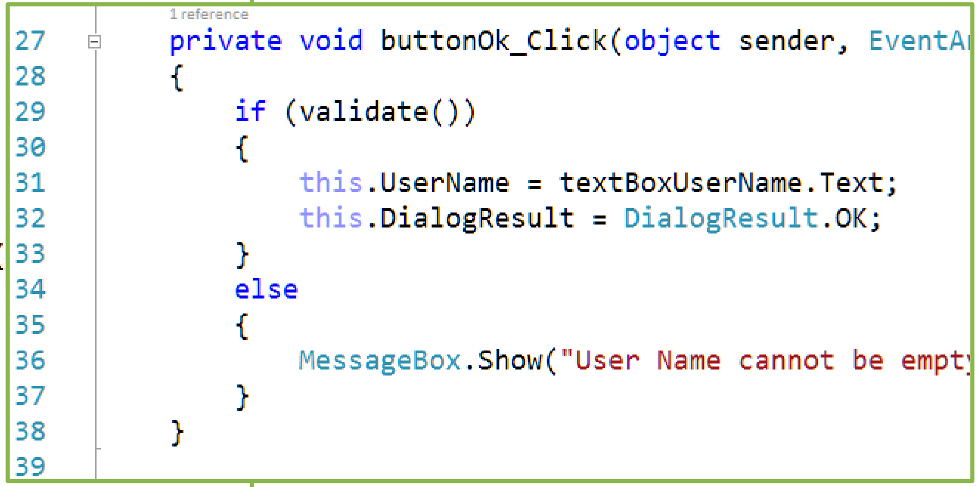
User Name: Cosmo Kramer

MainForm



```
1 reference
private void buttonShowEditDialog_Click(object sender,
{
    EditDialog dlg = new EditDialog();
    dlg.UserName = userName;
    dlg.ShowDialog();◄ Blocking Call

    if (dlg.DialogResult == DialogResult.OK
    {
        userName = dlg.UserName;
        labelResult.Text = userName;
    }

    dlg.Dispose();
```

Dialog



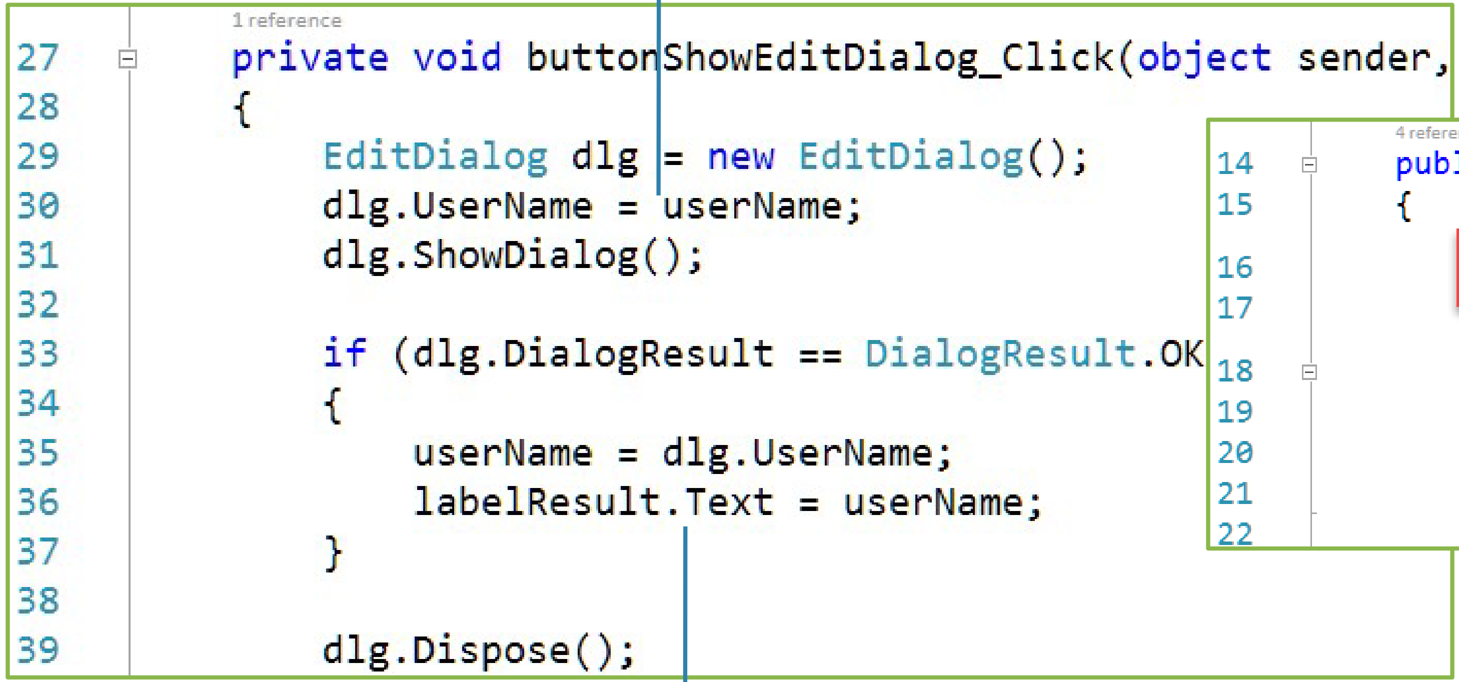
```
1 reference
private void buttonOk_Click(object sender, EventA
{
    if (validate())
    {
        this.UserName = textBoxUserName.Text;
        this.DialogResult = DialogResult.OK;
    }
    else
    {
        MessageBox.Show("User Name cannot be empt
    }
}
```

## Modal Dialogs – Passing Data Between Forms

Setting a Dialog Property from the MainForm

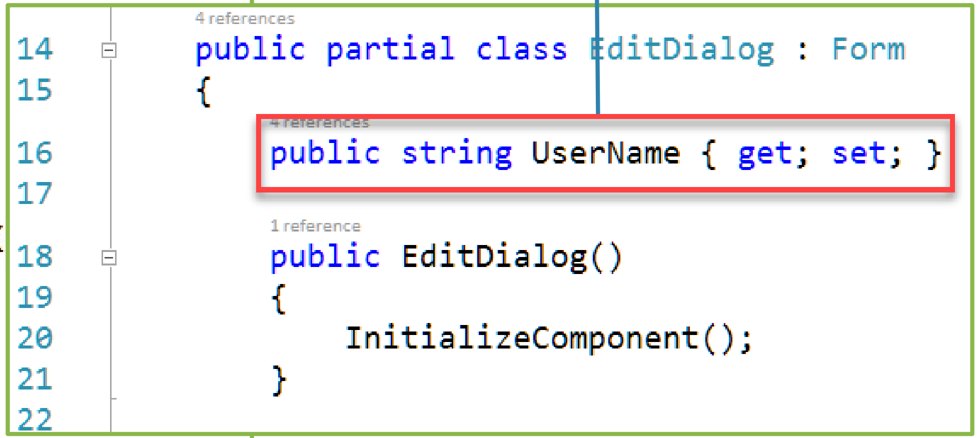
A Form is a class so you can expose private data through Property Methods

MainForm



```
1 reference
private void buttonShowEditDialog_Click(object sender,
{
    EditDialog dlg = new EditDialog();
    dlg.UserName = userName;
    dlg.ShowDialog();

    if (dlg.DialogResult == DialogResult.OK
    {
        userName = dlg.UserName;
        labelResult.Text = userName;
    }

    dlg.Dispose();
```

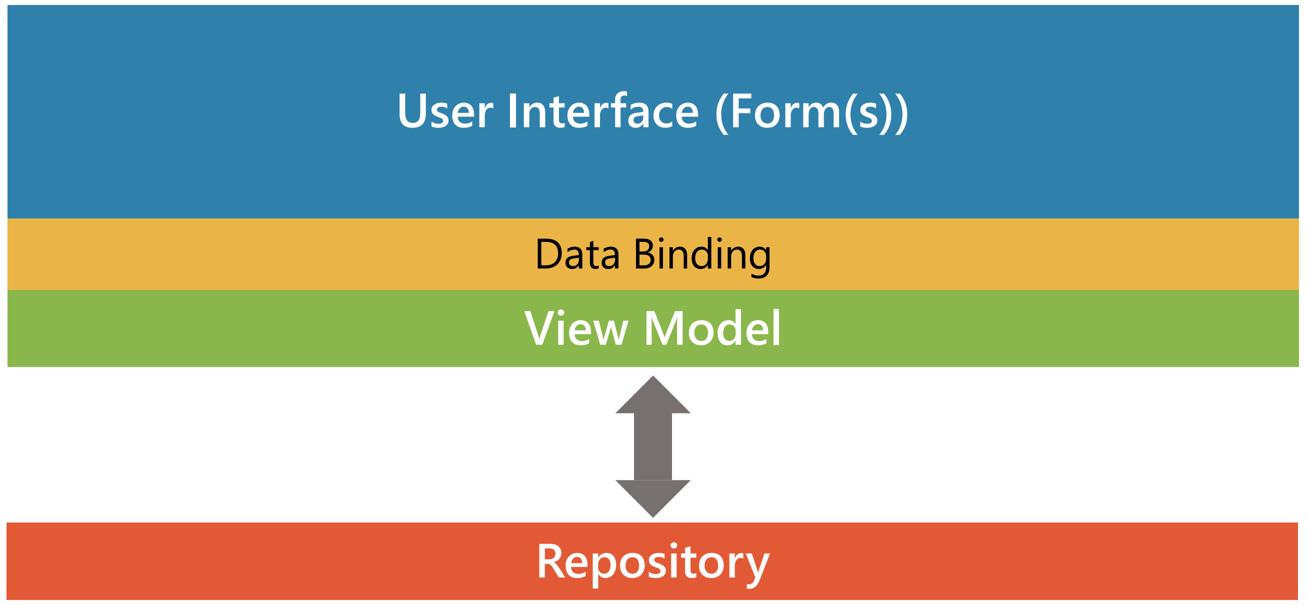
Dialog



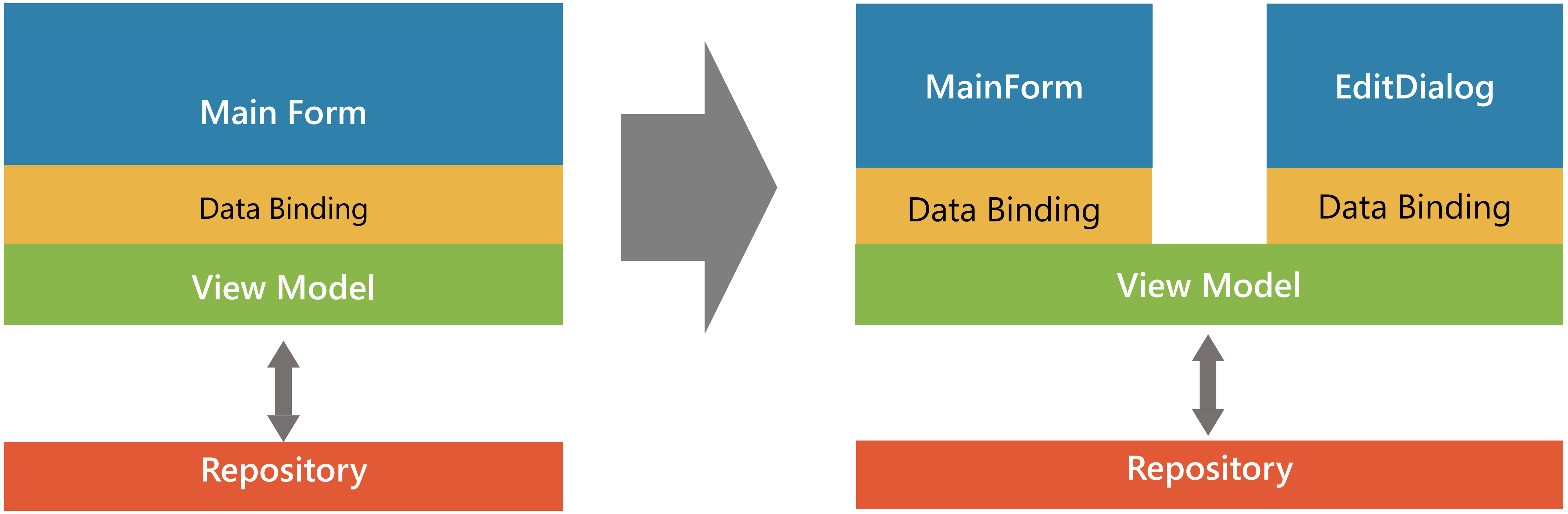
```
4 references
public partial class EditDialog : Form
{
    4 references
    public string UserName { get; set; }

    1 reference
    public EditDialog()
    {
        InitializeComponent();
    }
```

Reading a Dialog Property from the MainForm

## Assignment 7 Architecture (Part A)



## Assignment 7 Architecture (Part B)


Main Form
Data Binding
View Model
Repository
MainForm
EditDialog
Data Binding
Data Binding
View Model
Repository