

KEEPING IT CLASSY

WITH

Q#





Level 3

Groups of Objects

Create List of Musicians

Our bands will need to contain our musicians in order to store and announce them.

What our application will do:

- Store information about a band and it's musicians
- Announce the band
- Announce the musicians

In this level:

- Create a List of Musicians per band
- Create method to add a Musician to the List

Our Musician Object

We've created a simple Musician Class to hold data about our band's musicians.

Musician.cs

```
using System;

public class Musician
{
    public string Name;
    public string Instrument;

    public void Announce()
    {
        Console.WriteLine(Name + " on the " + Instrument + "!");
    }
}
```

Before we tackle groups of objects, we'll cover the short hand way to instantiate an object



Object_INITIALIZER

The Object_INITIALIZER allows us to instantiate and set variables in a single line.

Instantiating Musician without using Object_INITIALIZER

```
var musician = new Musician()  
Musician.Name = "Robert";  
Musician.Instrument = "Guitar";
```

Instantiating Musician with Object_INITIALIZER

```
Musician musician = new Musician{ Name="Robert", Instrument="Guitar" }
```



Between Curly Braces we set all our variables separating variables using commas

To add a group of Musicians to our Band class, we need to know how to group objects

Types of Groups of Objects

In C# we have several choices for grouping objects. Here are two examples:

Array

List

Arrays

Arrays are best used for a group of strongly-typed objects of a fixed number.

We create our Array of Musicians

Array

How many objects fit in our array

```
Musician[] musicians = new Musician[2];  
musicians[0] = new Musician{Name="Robert", Instrument="Guitar"}  
musicians[1] = new Musician{Name="Sarah", Instrument="Keyboard"}
```

If Thomas joins though we need to change the size of our band...

To change the Array size we have to recreate the entire array.

```
musicians = new Musician[3];  
musicians[0] = new Musician{Name="Robert", Instrument="Guitar"}  
musicians[1] = new Musician{Name="Sarah", Instrument="Keyboard"}  
musicians[2] = new Musician{Name="Thomas", Instrument="Drums"}
```



This is going to be a pain to manage with bands constantly changing members...

Lists

Lists are great for handling a group of strongly-typed objects of unknown or varying number.

List

The List's Data-Type

We create our List of Musicians

```
List<Musician> Musicians = new List<Musician>();  
Musicians.Add(new Musician{Name="Robert", Instrument="Guitar"});  
Musicians.Add(new Musician{Name="Sarah", Instrument="Keyboard"});
```

If Thomas joins we can just add him using Add

```
Musicians.Add(new Musician{Name="Thomas", Instrument="Drums"});
```

The List will resize automatically!



This will be much nicer for handling bands constantly changing members and won't break if we expand Musician later!

Musicians List

Add a using directive for System.Collections.Generic and change the Musician's variable.

Band.cs

```
using System;  
using System.Collections.Generic;  
  
public class Band  
{  
    public string Name;  
    public List<Musician> Musicians;  
  
    public void Announce() {...}  
}
```

List is in the built-in

*System.Collections.Generic
Namespace*

*Change Musicians to a List of **Musician***

*Now that we have our list of **Musician** objects, we should create a method to add to it*

Create AddMusician Method

Declare our new Musician variable that we'll add to our Musicians List.

Band.cs

```
...  
public void AddMusician()  
{  
    Musician musician = new Musician();  
}  
...  
  
OR  
  
[var] musician = new Musician();
```

var will infer our variable's datatype based on whatever is after the = character



The compiled results with or without var will be the same so do what you prefer!

Create AddMusician Method

Set musician's Name and Instrument variables with Console.ReadLine.

Band.cs

```
...
public void AddMusician()
{
    var musician = new Musician();
    Console.WriteLine("What is the name of the musician to be added?");
    musician.Name = Console.ReadLine();
    Console.WriteLine("What instrument does " + musician.Name + " play?");
    musician.Instrument = Console.ReadLine();
}
...
```


Create AddMusician Method

Add musician to our Musicians List using the Add method.

Band.cs

```
...  
public void AddMusician()  
{  
    var musician = new Musician();  
    Console.WriteLine("What is the name of the musician to be added?");  
    musician.Name = Console.ReadLine();  
    Console.WriteLine("What instrument does " + musician.Name + " play?");  
    musician.Instrument = Console.ReadLine();  
    Musicians.Add(musician);  
}  
...
```

Add will add the passed in musician object to the Musician's List

Error on musicians.Add

When we actually run our code we'll receive a `NullException` error on our `Add` method!



Band.cs

```
...
public void AddMusician()
{
    var musician = new Musician();
    Console.WriteLine("What is the name of the musician to be added?");
    musician.Name = Console.ReadLine();
    Console.WriteLine("What instrument does " + musician.Name + " play?");
    musician.Instrument = Console.ReadLine();
    Musicians.Add(musician);
}
...
```

If we look in the debugger we'll see Musicians is null!



ERROR: Object reference is not set to an instance of an object

Why is Musicians null? Shouldn't it be a List of Musician?

Null

Null is the absence of value. Not to be mistaken for zero as that is a value of nothing.

To the computer this says:

"You will be given a List of Musician; call it Musicians"

"Okay, Add musician to Musicians"



Band.cs

```
public class Band
{
    public List<Musician> Musicians;
    ...
    Musicians.Add(musician);
}
```

The computer responds:

"I don't understand! You only told me you were going to give me a list, you never actually gave it to me!"

Set a Default Value

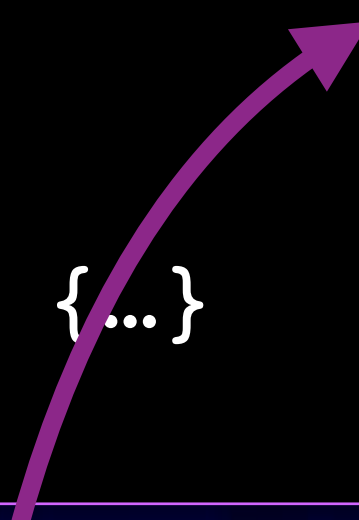
Set the Musicians variable to have a default value with `new List<Musician>()`.

Band.cs

```
using System;
using System.Collections.Generic;

public class Band
{
    public string Name;
    public List<Musician> Musicians = new List<Musician>();

    public void Announce() {...}
    public void AddMusician() {...}
}
```



This provides Musicians with initial empty List of Musician to work with

Musicians Internal Code Working

We added the underlying code for Musicians, we'll cover loops in the next level to wire it up!

Our Application now contains:

- A List of Musicians per band
- A method to add a Musician to the List

Example Running AddMusician Method

What is the name of the musician to be added?

>>>

\$ Robert

"What instrument does Robert play?"

>>>

\$ Guitar

A Quick Recap on Groups of Objects

Groups of Objects come in a variety of types which have their own pros and cons.

- Arrays work best with strongly-typed objects of fixed number
- Lists work best with strongly-typed objects of unknown or varying number
- You must reference System.Collections.Generic to use Lists