

Anonymous Types

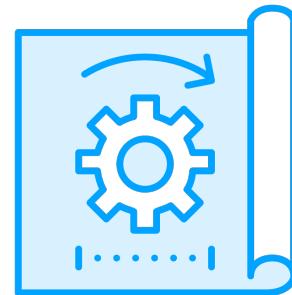


Filip Ekberg

Principal Consultant & CEO

@fekberg | fekberg.com

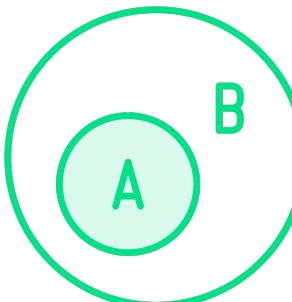
Anonymous Types



Temporary class
Use it to create a representation of data used in the current method



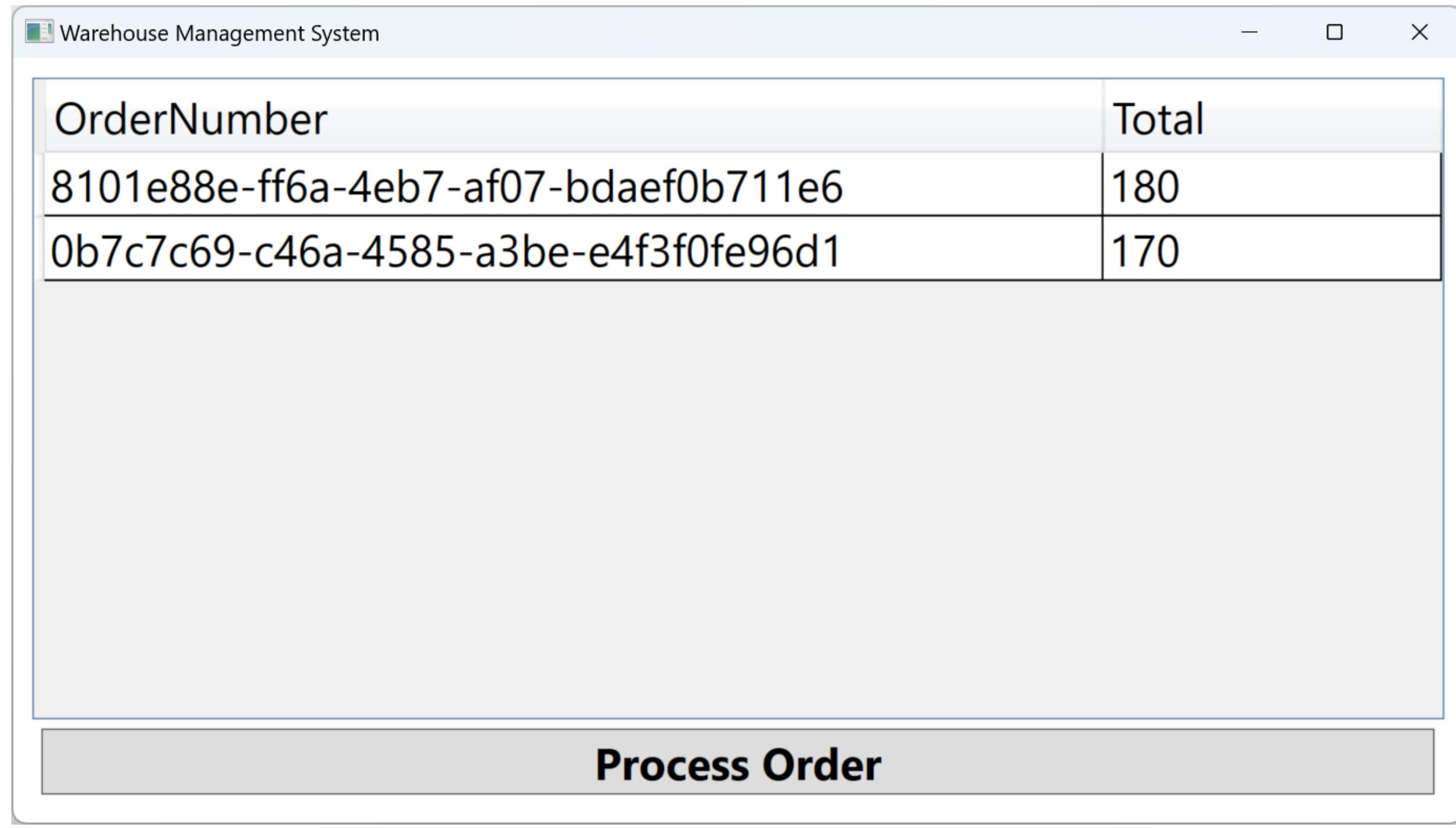
Do not return an anonymous type or use it as a parameter
It is only to be used in the current method



Commonly used with LINQ
Create a representation of a subset of the properties



List of Anonymous Type Bound to the UI



The screenshot shows a Windows application window titled "Warehouse Management System". Inside the window, there is a table with two columns: "OrderNumber" and "Total". The table contains two rows of data. A large, semi-transparent rectangular overlay covers most of the window's content area.

OrderNumber	Total
8101e88e-ff6a-4eb7-af07-bdaef0b711e6	180
0b7c7c69-c46a-4585-a3be-e4f3f0fe96d1	170

Process Order



Property names can be inferred by the compiler



**Explicitly define the names for
the properties that the
compiler cannot infer or where
a better name is appropriate**



Creating an Anonymous Type



Creating an Anonymous Type

```
var subset = new
{
    order.OrderNumber,
    order.Total,
    AveragePrice = order.LineItems.Average(item => item.Price)
};
```



Creating an Anonymous Type

```
var subset = new
{
    order.OrderNumber,
    order.Total,
    AveragePrice = order.LineItems.Average(item => item.Price)
};

var instance = new
{
    Read = new Func<string>(Console.ReadLine)
};

instance.Read();
```



Creating an Anonymous Type

```
var subset = new
{
    order.OrderNumber,
    order.Total,
    AveragePrice = order.LineItems.Average(item => item.Price)
};
```

```
var instance = new
{
    Read = new Func<string>(Console.ReadLine)
};
```

```
instance.Read();
```

This is not something you normally want to do!



Anonymous Types

“Anonymous types provide a convenient way to encapsulate a set of read-only properties into a single object without having to explicitly define a type first.”

The type name is generated by the compiler and is not available at the source code level.

The type of each property is inferred by the compiler.”



Using LINQ with Anonymous Types



Using LINQ with Anonymous Types

```
var totals = orders.Select(order => new { order.Total });
```



Using LINQ with Anonymous Types

```
// Method Syntax  
var totals = orders.Select(order => new { order.Total });
```



Using LINQ with Anonymous Types

```
// Method Syntax  
var totals = orders.Select(order => new { order.Total });  
  
// Query Syntax  
var totals = from order in orders  
            select new { order.Total };
```



Anonymous Type Bound to the UI

Order	Items	Total	IsReadyForShipment
0b7c7c69-c46a-4	2	170	<input checked="" type="checkbox"/>
8101e88e-ff6a-4	3	180	<input checked="" type="checkbox"/>

These **columns** were automatically generated based on the properties **available** on the **anonymous type**

Process Order



Creating an Anonymous Type

```
IEnumerable<Order> result = orders.Select(order => new
{
    order.OrderNumber,
    order.Total,
    AveragePrice = order.LineItems.Average(item => item.Price)
});
```



**Anonymous types are
read-only but support
non-destructive mutation**



Introducing the **with** Expression



Introducing the **with** Expression

```
var instance = new
{
    order.OrderNumber,
    order.Total,
    AveragePrice = order.LineItems.Average(item => item.Price)
};
```



Introducing the **with** Expression

```
var instance = new
{
    order.OrderNumber,
    order.Total,
    AveragePrice = order.LineItems.Average(item => item.Price)
};

var copy =
```



Introducing the **with** Expression

```
var instance = new
{
    order.OrderNumber,
    order.Total,
    AveragePrice = order.LineItems.Average(item => item.Price)
};

var copy = instance with { Total = 50 };
```



Introducing the **with** Expression

```
var instance = new
{
    order.OrderNumber,
    order.Total,
    AveragePrice = order.LineItems.Average(item => item.Price)
};
```

```
var copy = instance with { };
```



You **cannot add or remove properties!**
Only **specify which modifications** you want.



Introducing the **with** Expression

```
var instance = new
{
    order.OrderNumber,
    order.Total,
    AveragePrice = order.LineItems.Average(item => item.Price)
};

var copy = instance with { Total = 50 };
```



Immutability is beneficial in multi-threaded applications



**Only the reference is
copied for reference types!**



You shouldn't return an
anonymous type or use it as a
parameter!

But that doesn't mean you can't!
What happens if you do?



The Anonymous Type Is a Reference Type

```
object result = new { ... };
```



Constructing an Anonymous Type



Constructing an Anonymous Type

```
var instance = new  
{  
};
```



Constructing an Anonymous Type

```
var instance = new
{
    order.OrderNumber,
    order.Total,
    AveragePrice = order.LineItems.Average(item => item.Price)
};
```



Constructing an Anonymous Type

```
var instance = new  
{  
    order.OrderNumber,  
    order.Total,  
    AveragePrice = order.LineItems.Average(item => item.Price)  
};
```

The **compiler will generate an anonymous type** for you



**Don't return anonymous
types or use them as
parameters**



Using Reflection on Anonymous Type

```
var result = processor.Process(orders);

var type = result.GetType();
var properties = type.GetProperties();

foreach(var property in properties)
{
    Console.WriteLine($"Property: {property.Name}");
    Console.WriteLine($"Value: {property.GetValue(result)}");
}
```



Anonymous Type and LINQ

```
var summaries = orders.Select(order =>
{
    return new
    {
        Order = order.OrderNumber,
        Items = order.LineItems.Count(),
        Total = order.LineItems.Sum(item => item.Price),
        LineItems = order.LineItems
    };
}) ;
```



Up Next:

Tuples and Deconstruction

