Chapter 7: Null-Coalescing Operator

Parameter	Details
`possibleNullObject`	The value to test for null value. If non null, this value is returned. Must be a nullable type.
`defaultValue`	The value returned if `possibleNullObject` is null. Must be the same type as `possibleNullObject`.

Section 7.1: Basic usage

Using the `null-coalescing operator (??)` allows you to specify a default value for a nullable type if the left-hand operand is `null`.

```
string testString = null;
Console.WriteLine("The specified string is - " + (testString ?? "not provided"));
```

Live Demo on .NET Fiddle

This is logically equivalent to:

```
string testString = null;
if (testString == null)
{
    Console.WriteLine("The specified string is - not provided");
}
else
{
    Console.WriteLine("The specified string is - " + testString);
}
```

or using the ternary operator (?:) operator:

```
string testString = null;
Console.WriteLine("The specified string is - " + (testString == null ? "not provided" :
testString));
```

Section 7.2: Null fall-through and chaining

The left-hand operand must be nullable, while the right-hand operand may or may not be. The result will be typed accordingly.

Non-nullable

```
int? a = null;
int b = 3;
var output = a ?? b;
var type = output.GetType();

Console.WriteLine($"Output Type :{type}");
Console.WriteLine($"Output value :{output}");
```

Output:

```
Type :System.Int32
```

> value :3

View Demo

Nullable

```
int? a = null;
int? b = null;
var output = a ?? b;
```

`output` will be of type `int?` and equal to `b`, or `null`.

Multiple Coalescing

Coalescing can also be done in chains:

```
int? a = null;
int? b = null;
int c = 3;
var output = a ?? b ?? c;

var type = output.GetType();
Console.WriteLine($"Type :{type}");
Console.WriteLine($"value :{output}");
```

Output:

> Type :System.Int32
> value :3

View Demo

Null Conditional Chaining

The null coalescing operator can be used in tandem with the null propagation operator to provide safer access to properties of objects.

```
object o = null;
var output = o?.ToString() ?? "Default Value";
```

Output:

> Type :System.String
> value :Default Value

View Demo

Section 7.3: Null coalescing operator with method calls

The null coalescing operator makes it easy to ensure that a method that may return `null` will fall back to a default value.

Without the null coalescing operator:

```
string name = GetName();

if (name == null)
    name = "Unknown!";
```

With the null coalescing operator:

```
string name = GetName() ?? "Unknown!";
```

Section 7.4: Use existing or create new

A common usage scenario that this feature really helps with is when you are looking for an object in a collection and need to create a new one if it does not already exist.

```
IEnumerable<MyClass> myList = GetMyList();
var item = myList.SingleOrDefault(x => x.Id == 2) ?? new MyClass { Id = 2 };
```

Section 7.5: Lazy properties initialization with null coalescing operator

```
private List<FooBar> _fooBars;

public List<FooBar> FooBars
{
    get { return _fooBars ?? (_fooBars = new List<FooBar>()); }
}
```

The first time the property `.FooBars` is accessed the `_fooBars` variable will evaluate as `null`, thus falling through to the assignment statement assigns and evaluates to the resulting value.

Thread safety

This is **not thread-safe** way of implementing lazy properties. For thread-safe laziness, use the `Lazy<T>` class built into the .NET Framework.

C# 6 Syntactic Sugar using expression bodies

Note that since C# 6, this syntax can be simplified using expression body for the property:

```
private List<FooBar> _fooBars;

public List<FooBar> FooBars => _fooBars ?? ( _fooBars = new List<FooBar>() );
```

Subsequent accesses to the property will yield the value stored in the `_fooBars` variable.

Example in the MVVM pattern

This is often used when implementing commands in the MVVM pattern. Instead of initializing the commands eagerly with the construction of a viewmodel, commands are lazily initialized using this pattern as follows:

```
private ICommand _actionCommand = null;
public ICommand ActionCommand =>
   _actionCommand ?? ( _actionCommand = new DelegateCommand( DoAction ) );
```