<Coding𝄞onata />

12 Shorthand Operators Every C# Developer Should Know About

Aram Tchekrekjian

@AramT87

Conditional Operator ? :

Or, the Ternary Operator, it is a shortened and a cleaner way to write if-else statement.

```
string? blogName = "codingsonata";
string defaultBlogName = "Untitled Blog";

string displayName = string.IsNullOrEmpty(blogName) ? defaultBlogName : blogName;

Console.WriteLine($"Display name is {displayName}");
// Output: Display name is codingsonata
```

Null-Conditional Operator Member Access ?.

AKA. Null Propagator, added in C# 6, this is used to allow safe access for object properties, so a null object would just return a null instead of throwing an exception

```cs
public class BlogModel
{
    public int Id { get; set; }
    public string? Name { get; set; }
}

BlogModel? blogModel = null;

Console.WriteLine($"Display name is {blogModel?.Name}");
// Output: Display name is Untitled Blog
```

Null-Conditional Operator Element Access ?[]

This is similar to the previous operator, but used with Collections and Arrays rather than objects

```
public class BlogModel
{
    public int Id { get; set; }
    public string? Name { get; set; }
}

List<BlogModel>? blogModels = null;

Console.WriteLine($"Display name is {blogModels?[0].Name}");
// Output: Display name is
```

Null-Coalescing Operator ??

Existing since C# 2, it is used to combine the value of an operand with null to generate one value, which is either the value of the left-hand operand if it is not null, or it is the value of the right-hand operand otherwise.

```
string? displayName = null;
string? blogName = "codingsonata";
string defaultBlogName = "Untitled Blog";

displayName = blogName ?? defaultBlogName;
Console.WriteLine($"Display name is {displayName}");
//Output: Display name is codingsonata

blogName = null;
displayName = blogName ?? defaultBlogName;
Console.WriteLine($"Display name is {displayName}");
// Output: Display name is Untitled Blog
```

Chained Null-Coalescing Operator ?? ??

The order of execution is done from right to left, this is a very clean way to return a default value if there are multiple null reference types, on the same line.

```
string? displayName = null;
string? blogName = "codingsonata";
string defaultBlogName = "Untitled Blog";
string? blogDescription = "Coding Blog";

blogName = null;

displayName = blogName ?? blogDescription ?? defaultBlogName;
Console.WriteLine($"Display name is {displayName}");
// Output: Display name is Coding Blog

blogDescription = null;
displayName = blogName ?? blogDescription ?? defaultBlogName;
Console.WriteLine($"Display name is {displayName}");
// Output: Display name is Untitled Blog
```

Null-Coalescing & Null-Conditional Operators ?. ??

A very useful combination to check for null object as well the nullability of the accessed member with returning a default value if any of which is null

```
string? displayName = null;

BlogModel codingSonataBlog = new() { Id = 1, Name = "Coding Sonata" };

displayName = codingSonataBlog?.Name ?? defaultBlogName;

Console.WriteLine($"Display name is {displayName}");
// Output: Display name is Coding Sonata

BlogModel? unknownBlog = new() { Id = 2, Name = null };

displayName = unknownBlog?.Name ?? defaultBlogName;

Console.WriteLine($"Display name is {displayName}");
// Output: Display name is Untitled Blog
```

Null-Coalescing Assignment Operator ??=

Added in C# 8, it is similar to the Null-Coalescing operator, but can be used to assign the value of the left-operand with the value of the right-operand in case the left-operand was null

```
string? displayName = null;
BlogModel codingSonataBlog = new() { Id = 1, Name = "Coding Sonata" };

displayName ??= codingSonataBlog?.Name;

Console.WriteLine($"Display name is {displayName}");
// Output: Display name is Coding Sonata
```

Null-Forgiving Operator !

Or the Null-Suppression operator, introduced in C# 8.0
It is used to suppress a compile time null warning when you are sure the object won't be null

```
public class BlogModel
{
    public int Id { get; set; }
    public string? Name { get; set; }
}

BlogModel codingSonataBlog = new() { Id = 1, Name = "Coding Sonata" };

BlogModel? emptyBlogModel = null;

string anotherBlog = "Another Blog";

var blog = string.IsNullOrEmpty(anotherBlog) ? emptyBlogModel : codingSonataBlog;

Console.WriteLine($"Display name is {blog!.Name}");
// blog cannot be null in this case, so we can use null-forgiving operator to supress compile-time warning
// Output: Display name is Coding Sonata
```

Index Operator ^

Also called, index from end operator, introduced in C# 8, it is used to access element of a collection from the end of it rather than from the beginning

```
List<int> numberOfDays = Enumerable.Range(1, 30).ToList();

int lastDay = numberOfDays[^1];

Console.WriteLine($"Last Day is {lastDay}");
//Output: Last Day is: 30
```

Range Operator ..

Another operator introduced in C# 8, the range operator can be used to slice or take a range of values from any part of an array.

It can be used in combination with the Index Operator.

Note that the collections like list do not support the range operator, yet you can either write an extension method to implement it,

Or you can use the List.GetRange method to achieve the same result

Range and Index Operators in C#

.. ^

Introduced in C# 8 to provide flexible operators to work with collections, both operators can be used separately or combined

```
int[] arrayOfNumbers = Enumerable.Range(1, 15).ToArray();

int[]? firstSevenNumbers = arrayOfNumbers[..7];
// start from index 0 (number 1) and take up to 7 elements

Console.WriteLine($"Numbers are {string.Join(",", firstSevenNumbers)}");
// Output: Numbers are 1,2,3,4,5,6,7

int startIndex = 9;
int[]? last5Numbers = arrayOfNumbers[startIndex..^0];
// start from index 9 (number 10) and take up to last element (number 15)

Console.WriteLine($"Numbers are {string.Join(",", last5Numbers)}");
// Output: Numbers are 10,11,12,13,14,15
```

Expression Body Type Definition =>

Introduced in C# 6, this is a shorthand way to define the implementation of a function, property, constructor and others, with a single expression

```
public class BlogModel
{
    public int Id { get; set; }
    public string? Name { get; set; }
    public string? URL { get; set; }
    public string? Title
        => $"{(string.IsNullOrEmpty(Name) ? "Blog" : Name)} -" +
           $"{(string.IsNullOrEmpty(URL) ? "No link" : URL)}";
}

BlogModel blog = new() { Name = "CodingSonata", URL = "codingsonata.com" };

Console.WriteLine($"Blog title is {blog.Title}");
// Output: Blog title is CodingSonata - codingsonata.com
```

Type Testing Operator is

The is operator does a runtime type-check for expressions, it supports implicit boxing/unboxing without having to use cast.

From C#7, you can use the is operator with pattern matching, like property, constant, var

And starting from C# 9, you can start using the is operator in declaration, type, relational pattern matching

Slide next to see the sample code:

Type Testing Operator is

```
public class BlogModel
{
    public int Id { get; set; }
    public string? Name { get; set; }
}

BlogModel? someBlog = null;
if (someBlog is null || someBlog.Id is 0)
{
    Console.WriteLine($"No blog was found");
}
```

Type Testing Negation Operator is not

Introduced in C# 9, this shorthand operator can be used to negate a pattern matching, which is simply the negation of the is operator

```
public class BlogModel
{
    public int Id { get; set; }
    public string? Name { get; set; }
}

BlogModel? myBlog = new() { Id = 1, Name = "CodingSonata", URL = "codingsonata.com" };

if (myBlog is not null && myBlog.Id is not 0)
{
    Console.WriteLine($"Blog found with name {myBlog.Name}");
}
```

Type-casting Operator as

Converts a value to a reference type or nullable value type. If the conversion isn't possible, a null value will be returned.

Unlike when using a cast expression, if the conversion isn't possible, an exception will be thrown at runtime

See the next slide for examples:

Type-casting Operator as

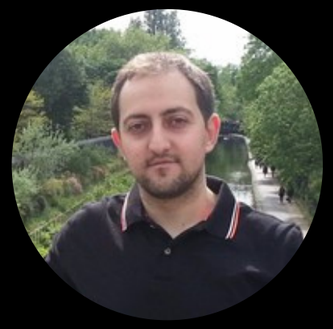
```
object boolean = false;
var operatorAsResult = boolean as string;

Console.WriteLine($"operatorAsResult is {operatorAsResult}");

var castExpressionResult = (string)boolean;
//System.InvalidCastException:
//'Unable to cast object of type 'System.Boolean' to type 'System.String'.'

Console.WriteLine($"castExpressionResult is {castExpressionResult}");
```

Found this Useful?

Consider Reposting

Thank You

Follow me for more content



Aram Tchekrekjian

Microsoft® MVP Most Valuable Professional

AramT87

CodingSonata.com/newsletters

<Coding Sonata />