Are you using C# like it's 2004 or 2024?

If you are falling behind on the latest C# updates, don't worry.

You are not alone.

It’s hard to keep up with all the latest additions to the C#.

I will show 7 features introduced in versions C# 10 - C# 12 I find the most useful.

#1 - File scoped namespace

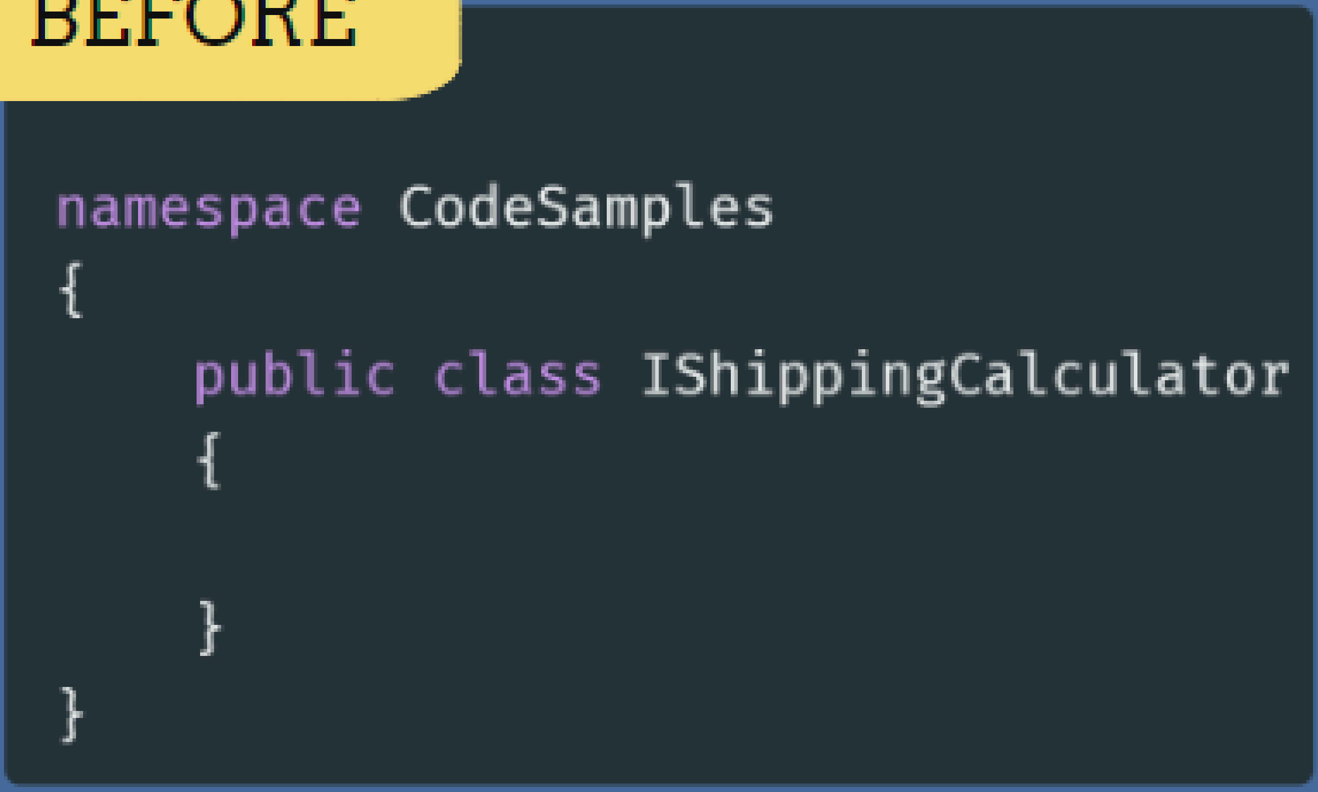
You can reduce nesting in your code by converting your class namespace to a file-scoped namespace.

This option removes top curly braces from your file.

Result: you remove one level of nesting.
And add one level of readability.

#1 - File scoped namespace

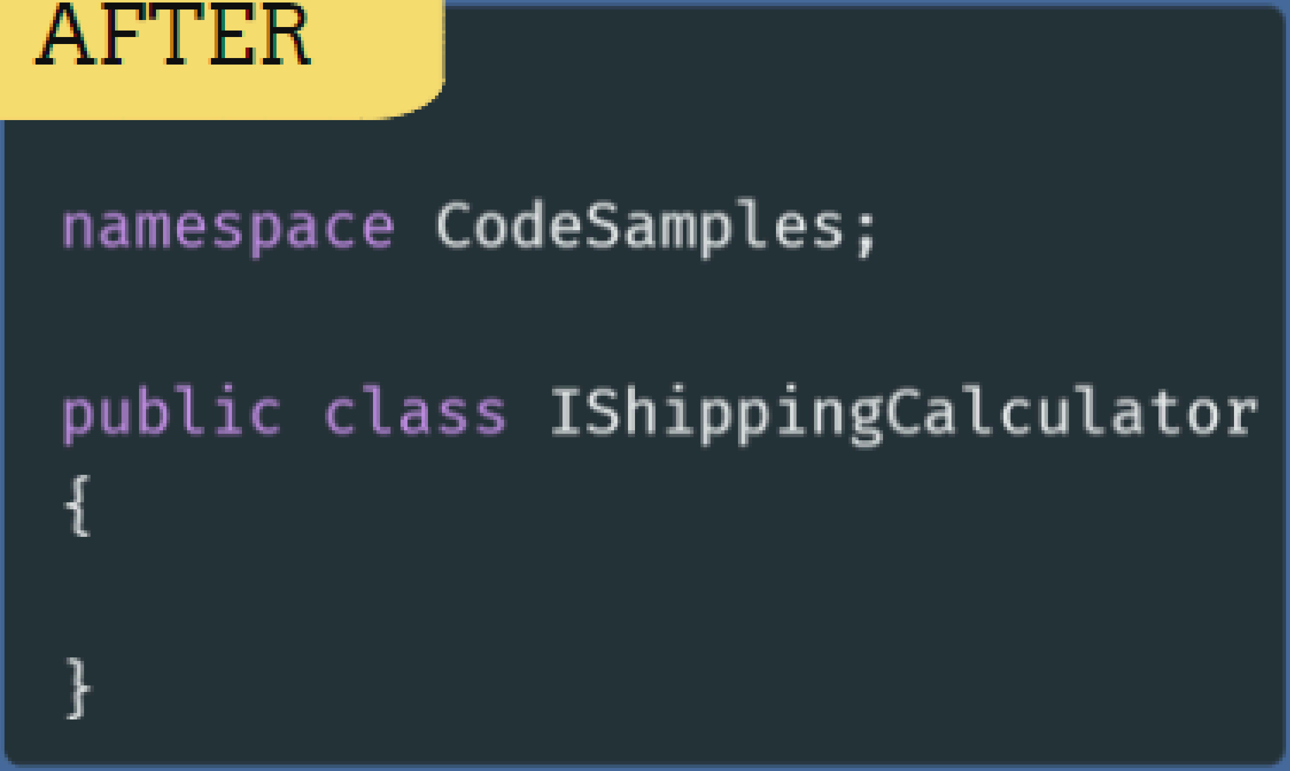
BEFORE



```
namespace CodeSamples
{
    public class IShippingCalculator
    {

    }
}
```

AFTER



```
namespace CodeSamples;

public class IShippingCalculator
{

}
```

#2 - Record struct

This feature adds a record struct option to the record types.

What are records?

A record in C# is a special type whose primary role is storing data.

Its 2 main characteristics are:

1. Value equality
2. Immutability

Therefore, records are a good option for DTOs.

#2 - Record struct

```csharp
public record ClassResponse(long Id, string Name,
                            string Description, int DefaultCapacity);

public record CreateClassRequest(string Name,
                                 string Description, int DefaultCapacity);

public record UpdateClassRequest(string Name,
                                 string Description, int DefaultCapacity);
```

#3 - Global using directive

If you have a namespace that you include regularly in most of your files, you can define it using the global directive.

Then that namespace will be available in all files inside your project.

```
global using <fully-qualified-namespace>;

//Example
global using System.Linq;
```

#4 - Raw string literals

With C# 11, you can define a string with any text, including:

- whitespace,
- new lines,
- quotes,
- other special characters,

without requiring escape characters.

To use it, define three double quotes at the start and the end of the string.

#4 - Raw string literals

```
string longMessage = """
    This is a long message.
    It has several lines.
        Some are indented
                more than others.
    Some should start at the first column.
    Some have "quoted text" in them.
    """;
```

#5 - List patterns

The list patterns feature allows you to compare the items in a list or array using various matching syntaxes.

For example, you can use the *is* keyword to do different comparisons against an array.

```
int[] numbers = { 1, 2, 3 };

Console.WriteLine(numbers is [1, 2, 3]);  // True
Console.WriteLine(numbers is [1, 2, 4]);  // False
Console.WriteLine(numbers is [1, 2, 3, 4]);  // False
Console.WriteLine(numbers is [0 or 1, ⩽ 2, ⩾ 3]);  // True
```

#6 - Primary constructors

Primary constructors represent a concise syntax to create constructors whose parameters are available anywhere within the class.

There is one catch, though.

You should treat them more as parameters to your class than as injected dependencies

#6 - Primary constructors

1. Primary constructor parameters aren’t members of the class.

2. Parameters are not read-only.

3. They don’t automatically become class properties, except in record types.

```
public interface IDistanceCalculator
{
    Distance GetDistance();
}

public class ExampleController(IDistanceCalculator service) : ControllerBase
{
    [HttpGet]
    public ActionResult<Distance> Get()
    {
        return service.GetDistance();
    }
}
```

#7 - Collection expressions

Collection expressions are a simpler way to create new instances of lists and arrays.

```csharp
// Create an array:
int[] a = [1, 2, 3, 4, 5, 6, 7, 8];

// Create a list:
List<string> b = ["one", "two", "three"];
```

→

Thanks for reading!

Please repost to help others learn the latest C# features.