Key Features review

C#

7

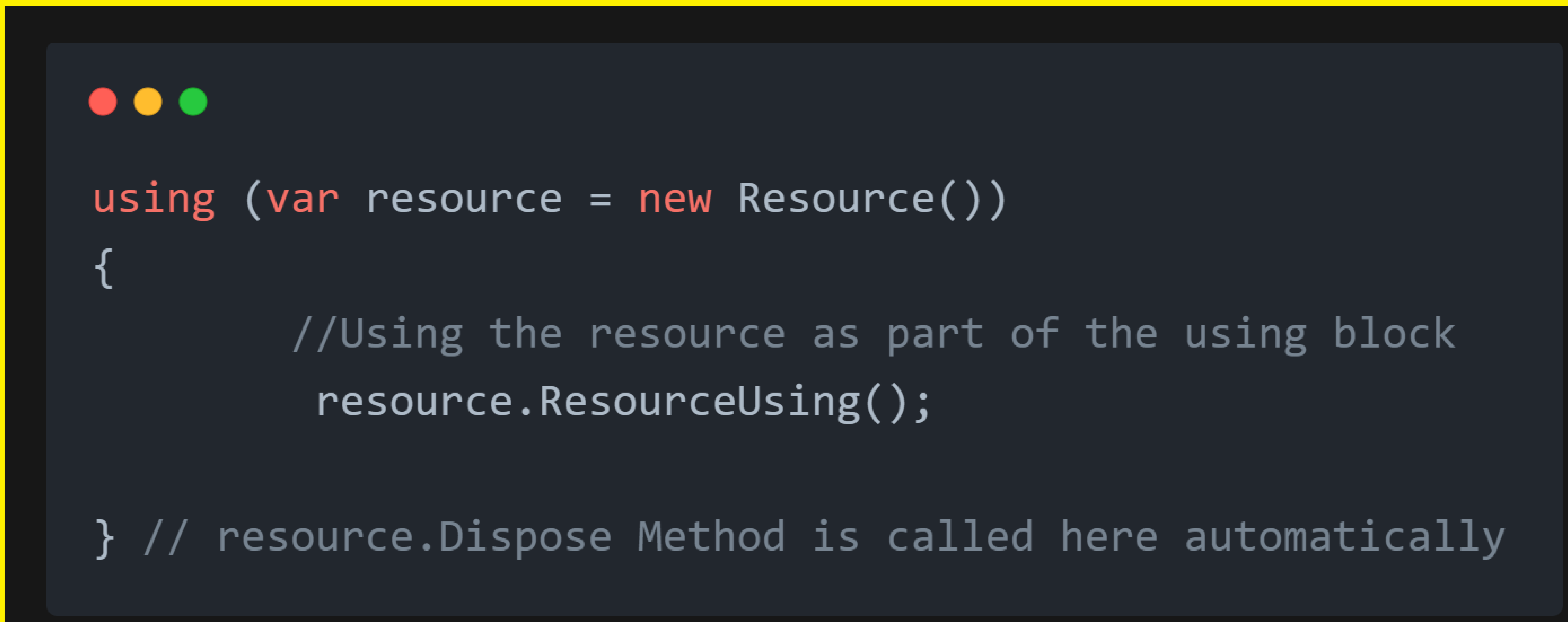
8

9

10

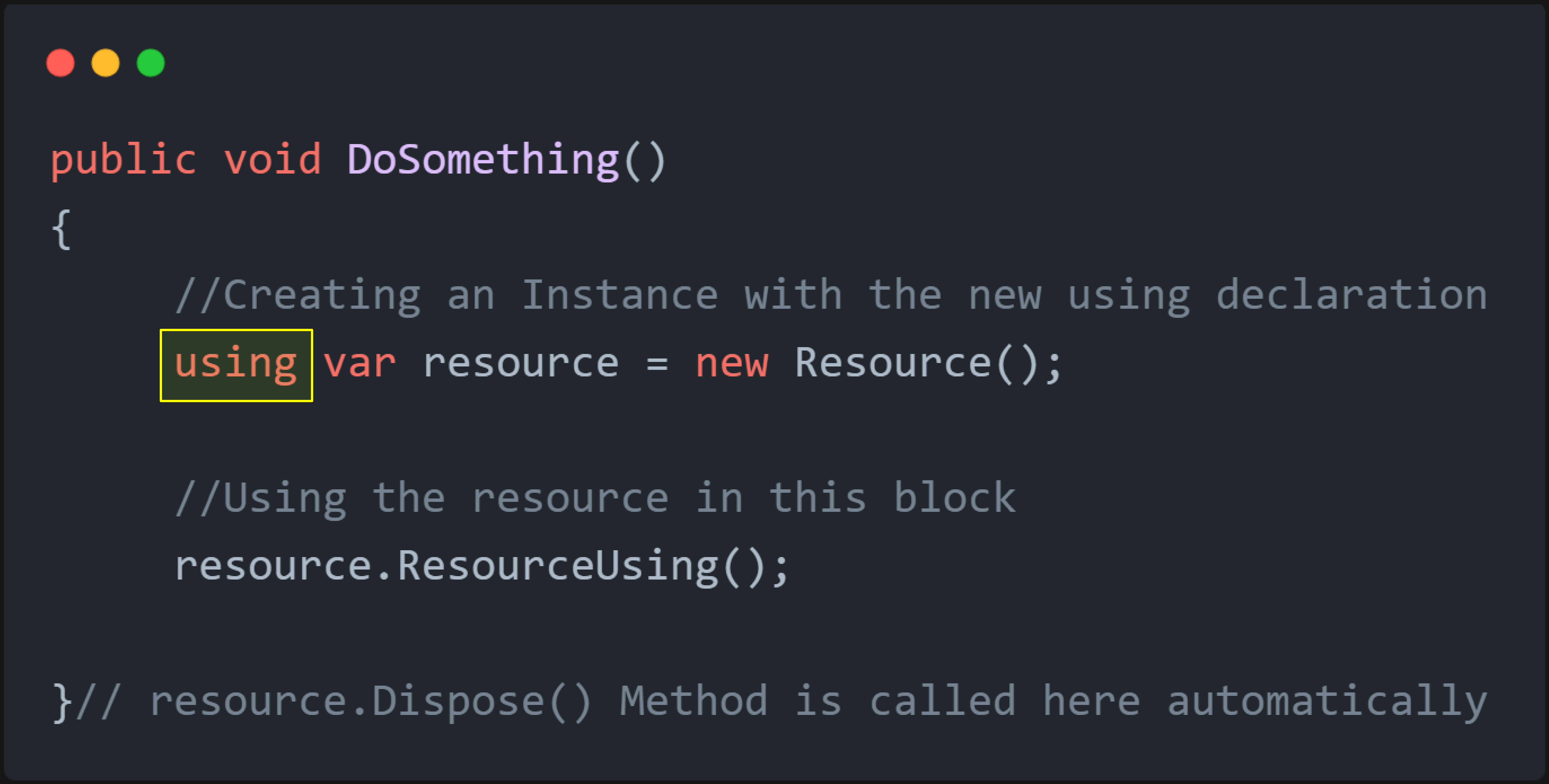
Nabi Karampoor
@thisisnabi

1 Using Declaration



```
using (var resource = new Resource())
{
        //Using the resource as part of the using block
         resource.ResourceUsing();

} // resource.Dispose Method is called here automatically
```

This is only available inside brackets and can be confusing if you have a lot of using.



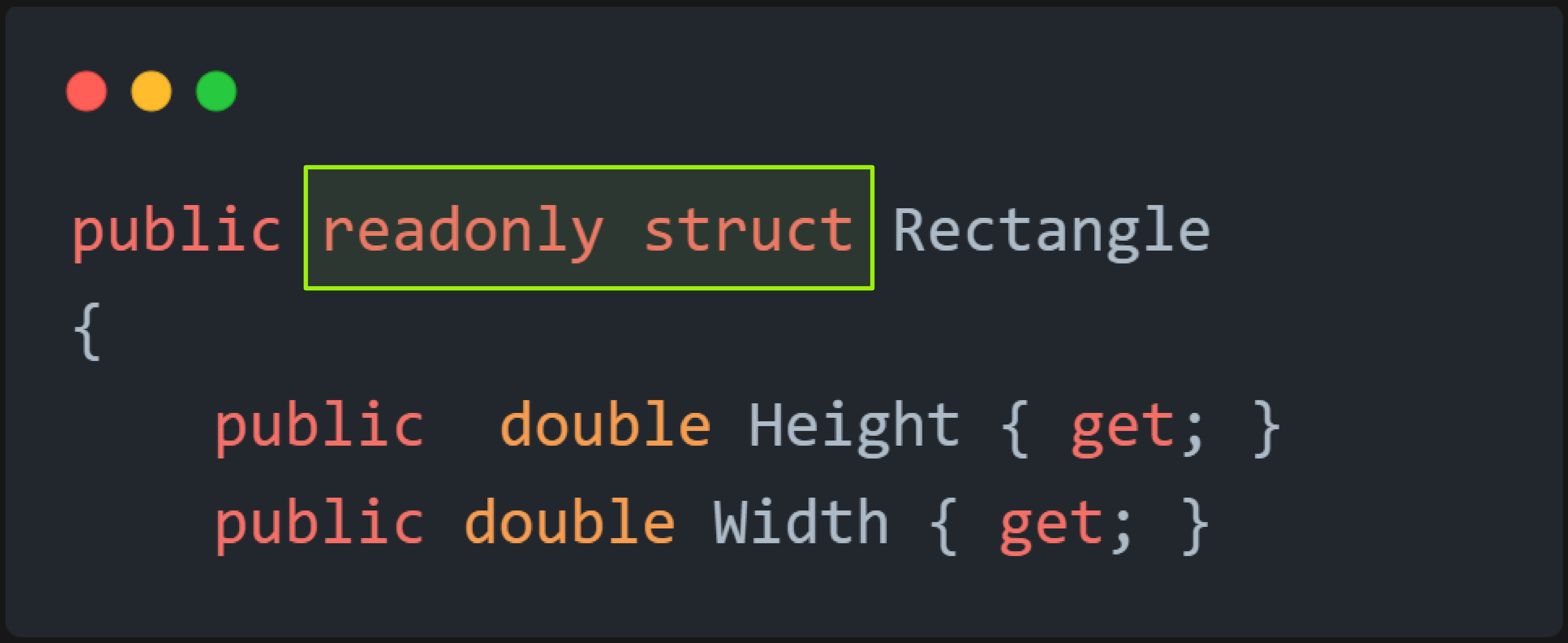
```
public void DoSomething()
{
     //Creating an Instance with the new using declaration
     using var resource = new Resource();

     //Using the resource in this block
     resource.ResourceUsing();

}// resource.Dispose() Method is called here automatically
```

Now, the brackets are no longer required

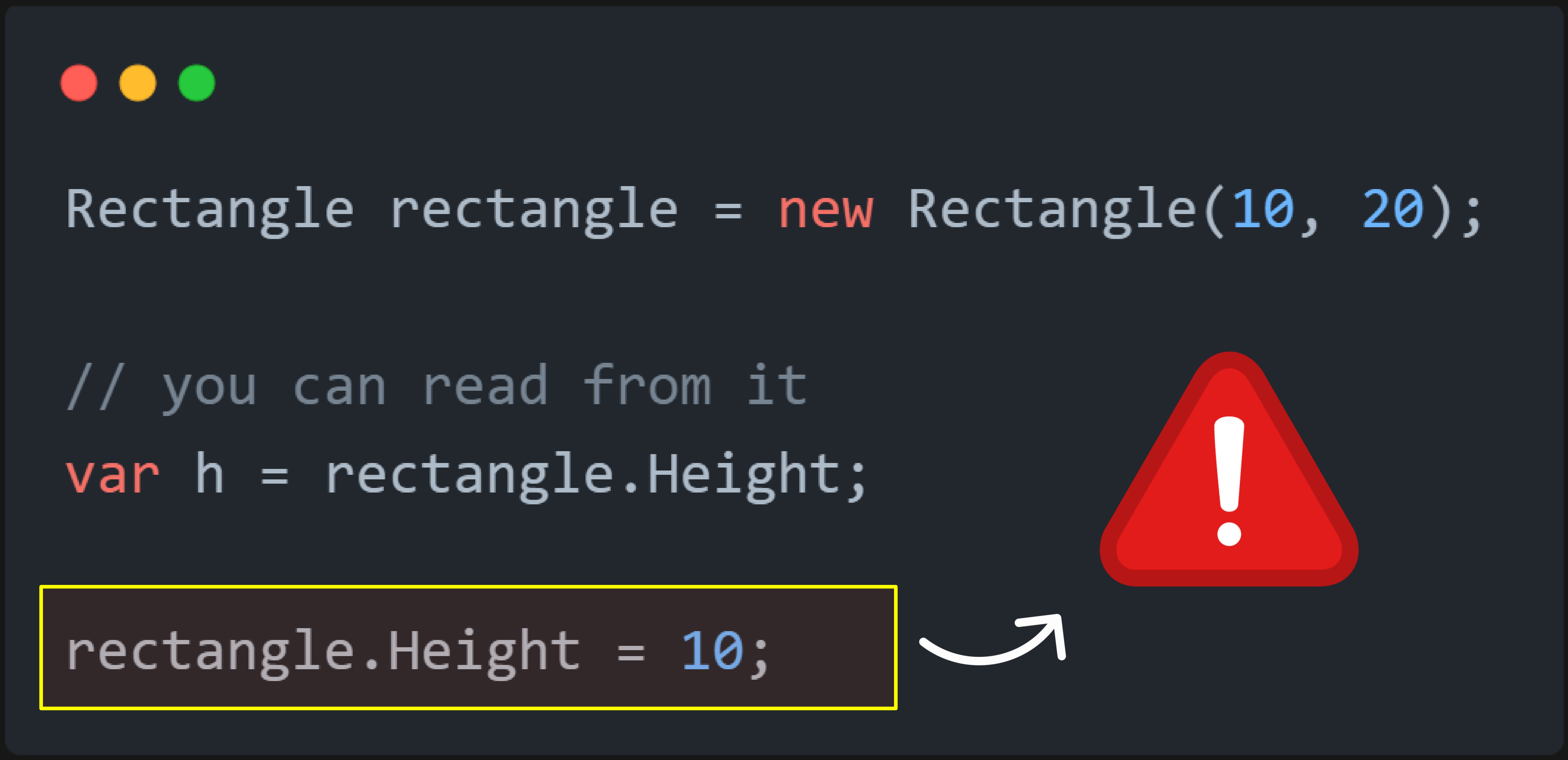
Nabi Karampoor

@thisisnabi

2 Read-only Struct



```
public readonly struct Rectangle
{
    public  double Height { get; }
    public double Width { get; }
```



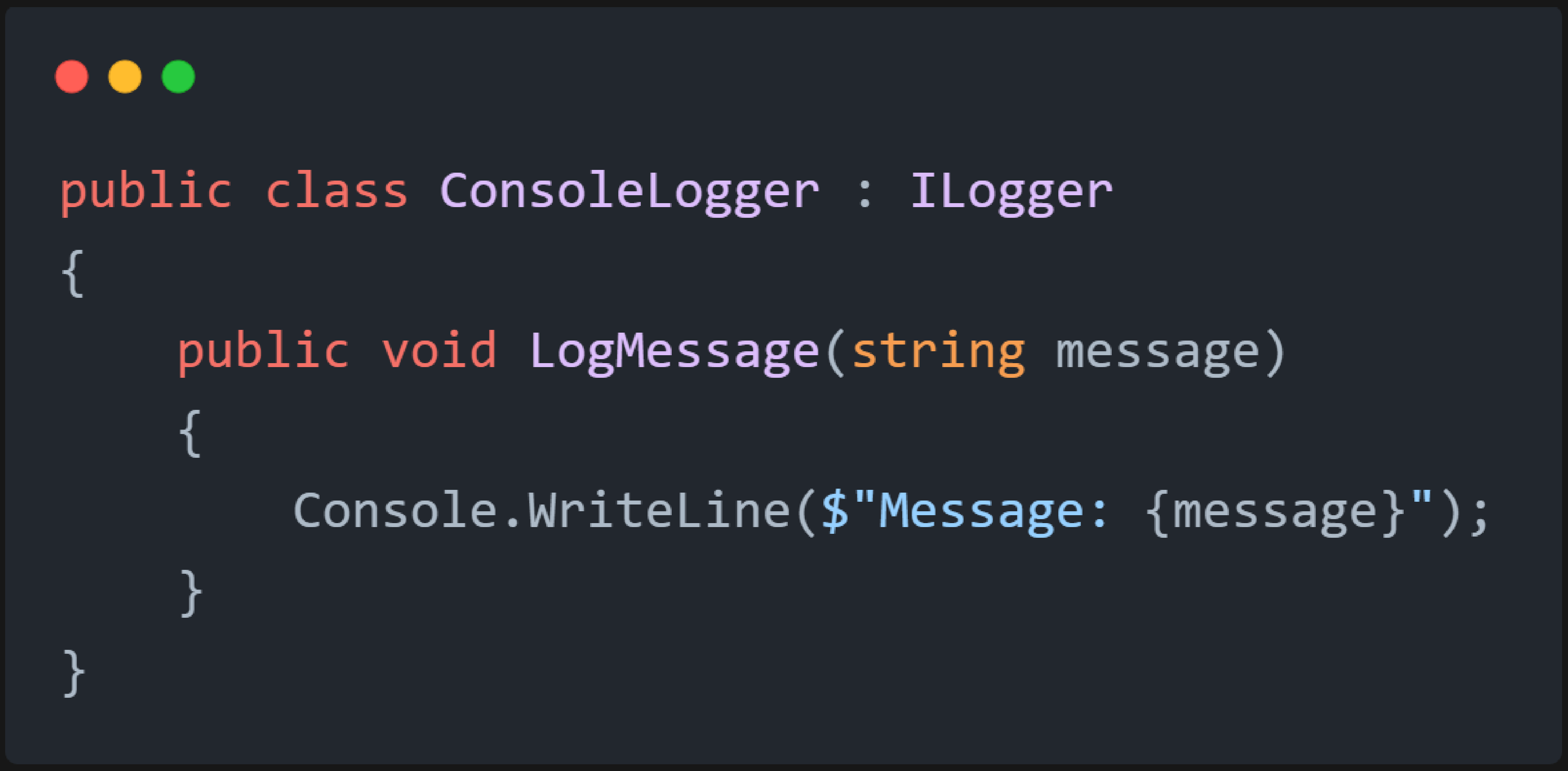
```
Rectangle rectangle = new Rectangle(10, 20);

// you can read from it
var h = rectangle.Height;

rectangle.Height = 10;
```

Provide immutability and enhanced performance by ensuring that instances cannot be modified after creation.

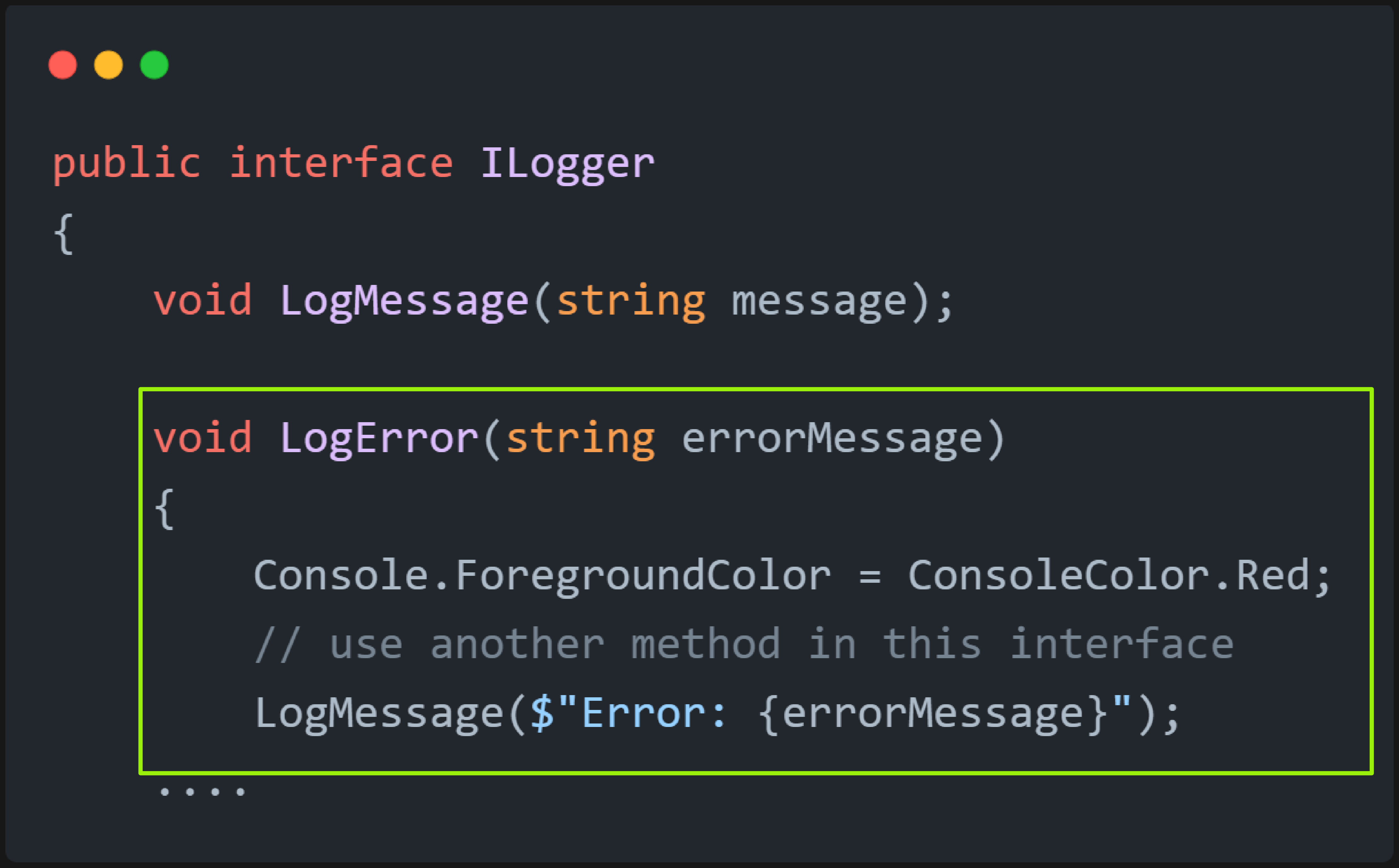
Nabi Karampoor
@thisisnabi

3 Default interface Methods

Backward compatibility

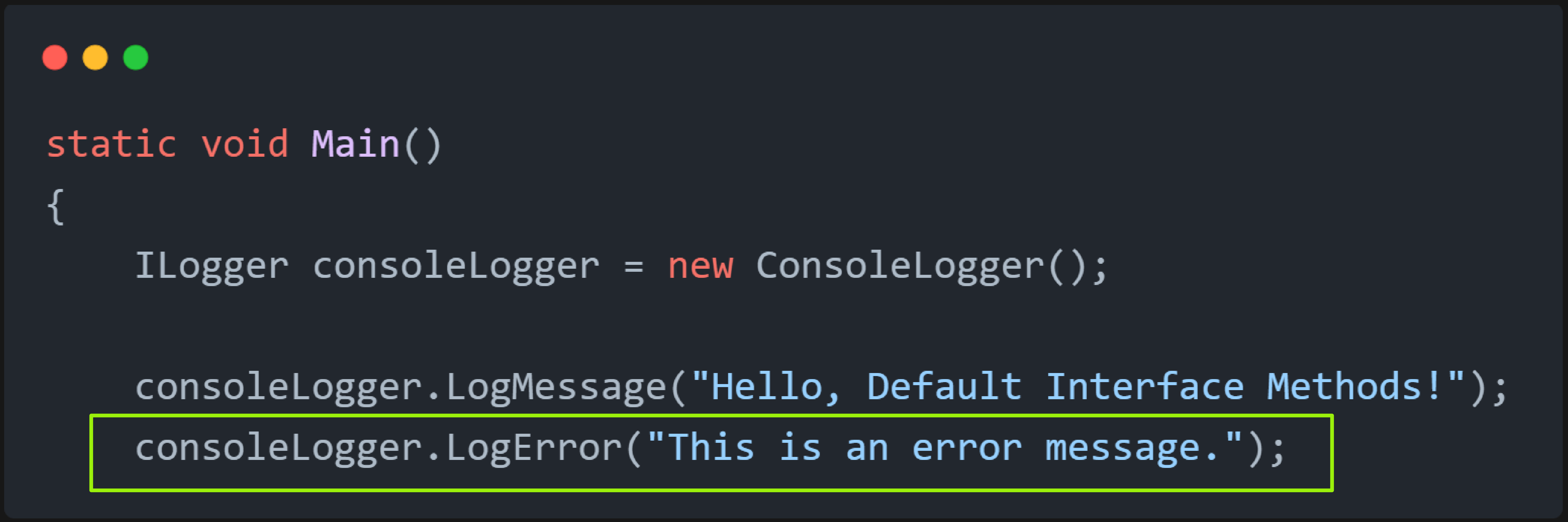


```
public class ConsoleLogger : ILogger
{
    public void LogMessage(string message)
    {
        Console.WriteLine($"Message: {message}");
    }
}
```



```
public interface ILogger
{
    void LogMessage(string message);

    void LogError(string errorMessage)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        // use another method in this interface
        LogMessage($"Error: {errorMessage}");
    ....
```

Adding new methods to interfaces without breaking existing implementing classes

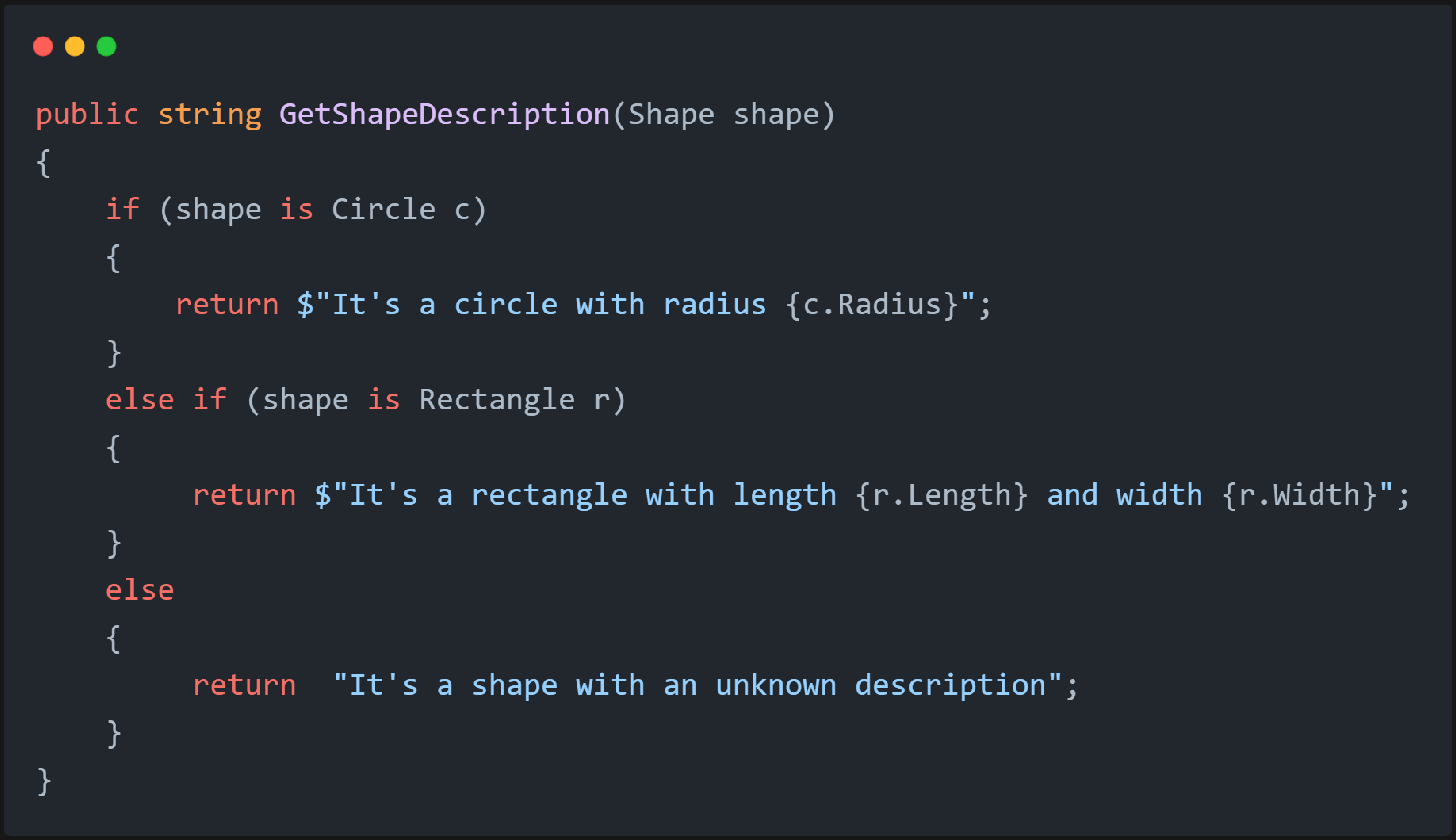


```
static void Main()
{
    ILogger consoleLogger = new ConsoleLogger();

    consoleLogger.LogMessage("Hello, Default Interface Methods!");
    consoleLogger.LogError("This is an error message.");
```

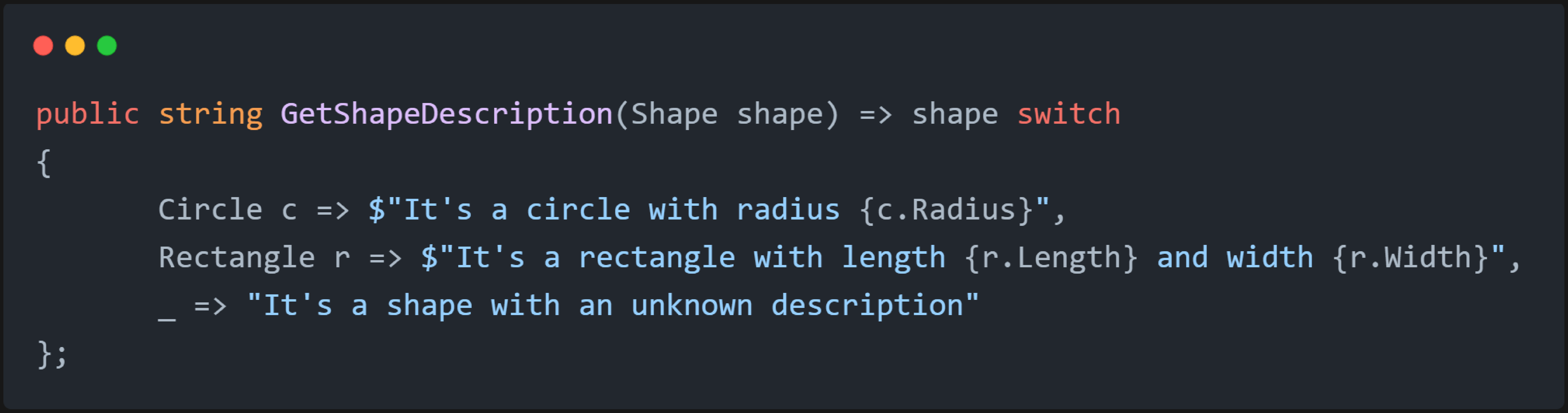
Nabi Karampoor
@thisisnabi

Switch Expressions

Enhanced Pattern Matching



```
public string GetShapeDescription(Shape shape)
{
    if (shape is Circle c)
    {
        return $"It's a circle with radius {c.Radius}";
    }
    else if (shape is Rectangle r)
    {
         return $"It's a rectangle with length {r.Length} and width {r.Width}";
    }
    else
    {
         return  "It's a shape with an unknown description";
    }
}
```



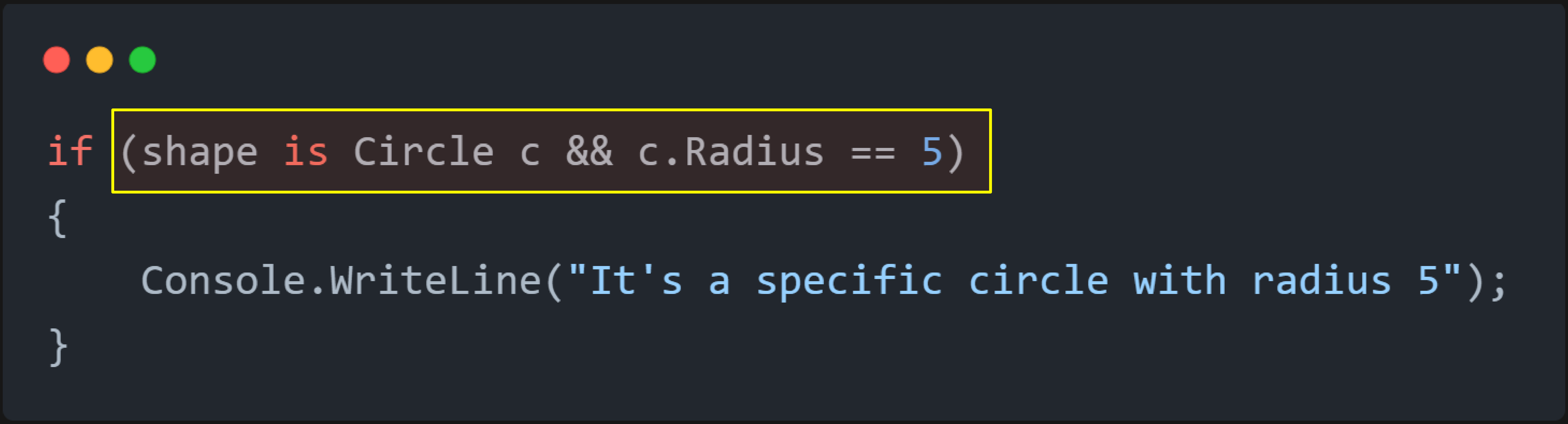
```
public string GetShapeDescription(Shape shape) => shape switch
{
       Circle c => $"It's a circle with radius {c.Radius}",
       Rectangle r => $"It's a rectangle with length {r.Length} and width {r.Width}",
       _ => "It's a shape with an unknown description"
};
```

Nabi Karampoor
@thisisnabi

4
2

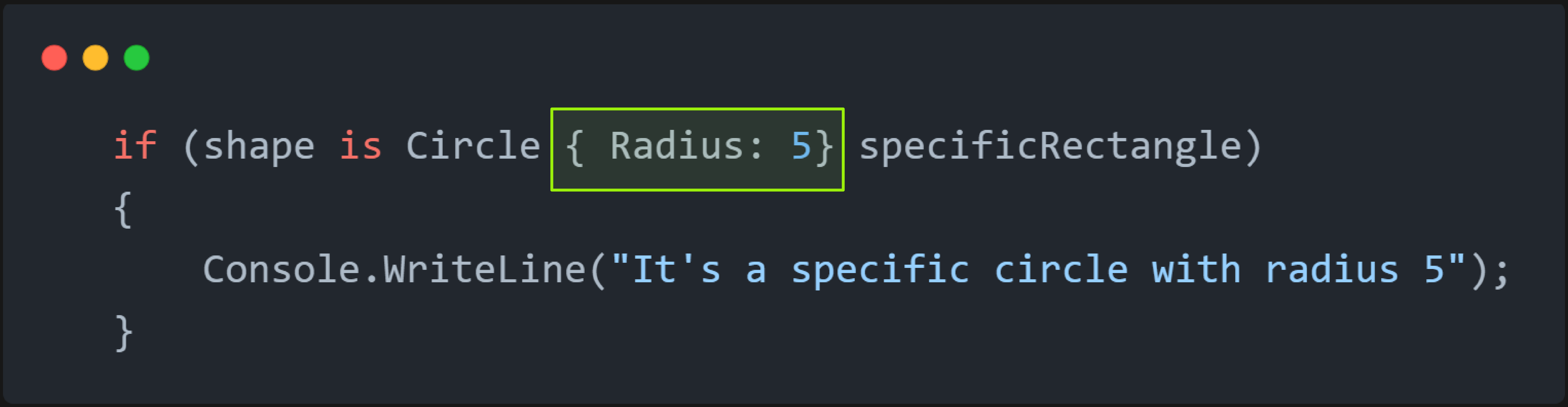
Property Patterns

Enhanced Pattern Matching



```
if (shape is Circle c && c.Radius == 5)
{
    Console.WriteLine("It's a specific circle with radius 5");
}
```

Checking and comparing the values of properties



```
if (shape is Circle { Radius: 5} specificRectangle)
{
    Console.WriteLine("It's a specific circle with radius 5");
}
```

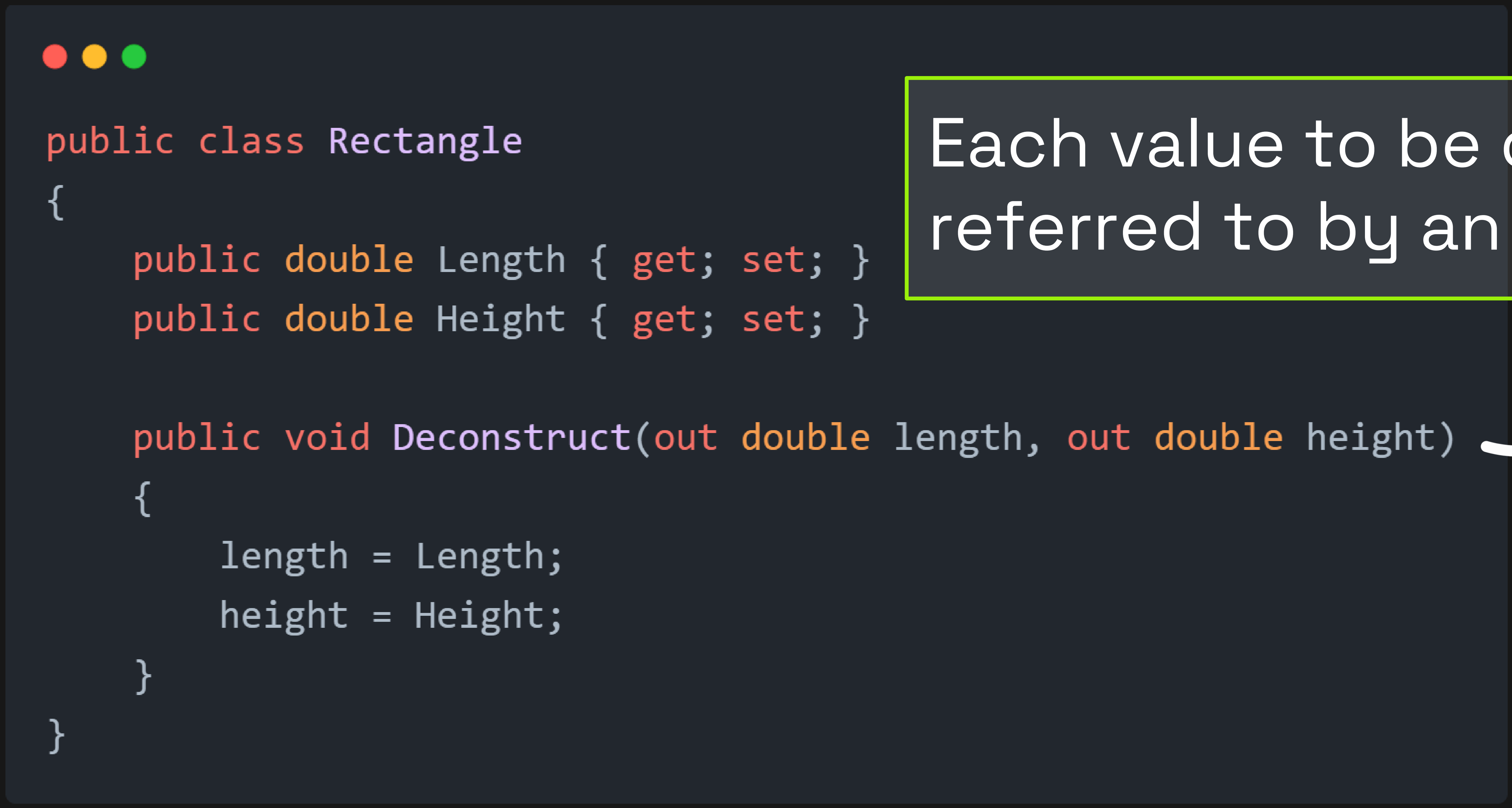
I think this is readable when the number of comparisons is small.

Nabi Karampoor
@thisisnabi

4
3

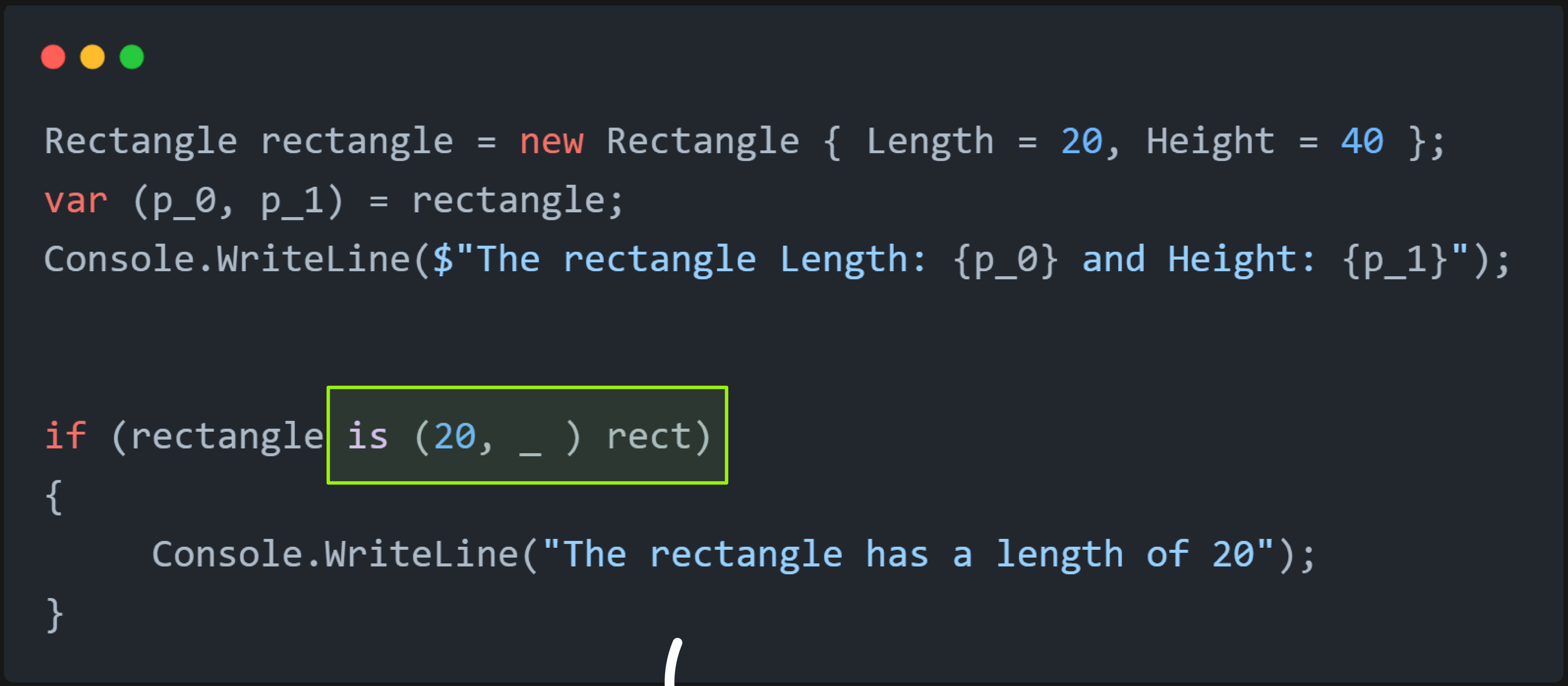
Positional Patterns

Enhanced Pattern Matching



```
public class Rectangle
{
    public double Length { get; set; }
    public double Height { get; set; }

    public void Deconstruct(out double length, out double height)
    {
        length = Length;
        height = Height;
    }
}
```

Each value to be deconstructed is referred to by an out parameter.



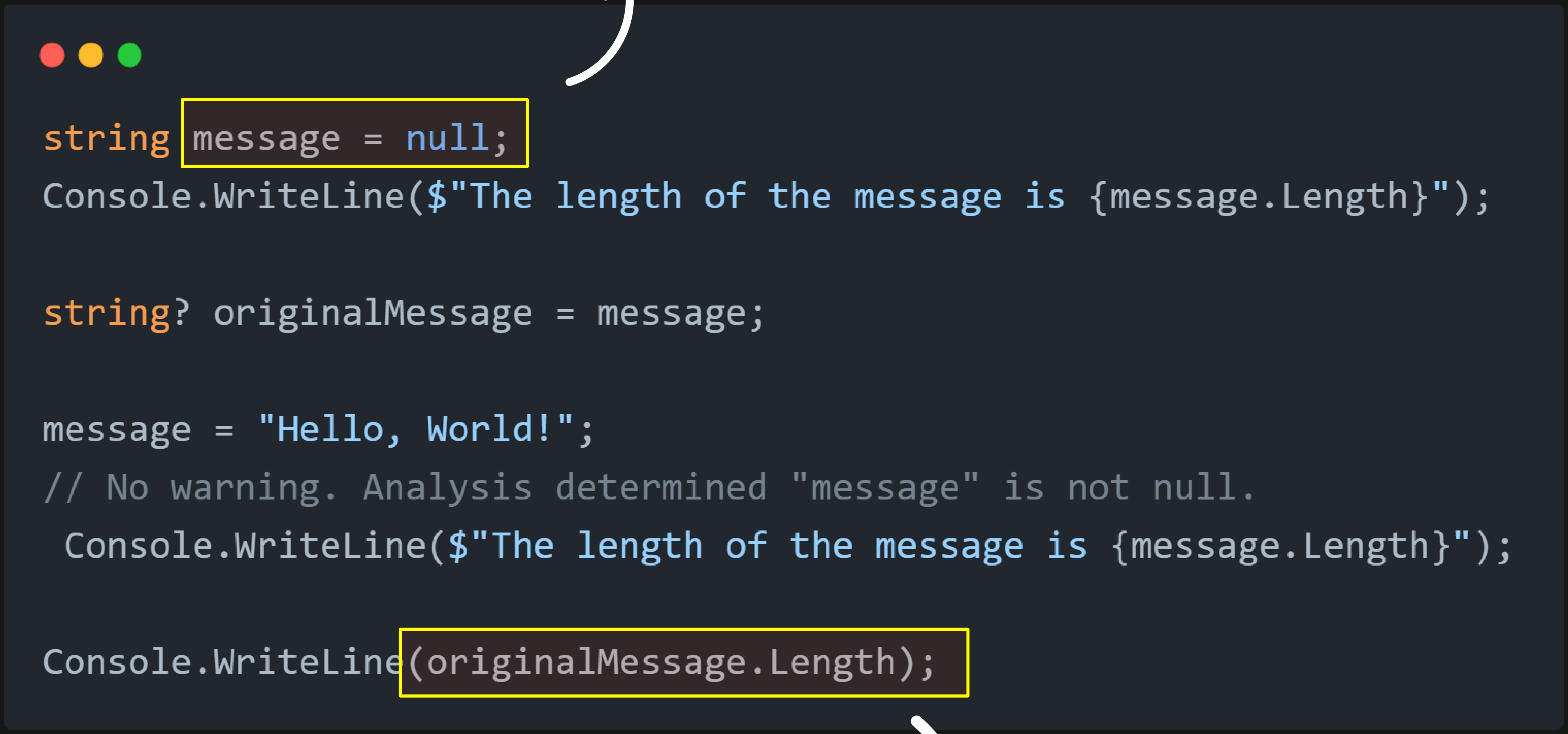
```
Rectangle rectangle = new Rectangle { Length = 20, Height = 40 };
var (p_0, p_1) = rectangle;
Console.WriteLine($"The rectangle Length: {p_0} and Height: {p_1}");


if (rectangle is (20, _ ) rect)
{
     Console.WriteLine("The rectangle has a length of 20");
}
```

_ Discards can be used where any value

Nabi Karampoor
@thisisnabi

5 Nullable Reference Types

Emits a compiler warning if a variable that must not be null is assigned to null.



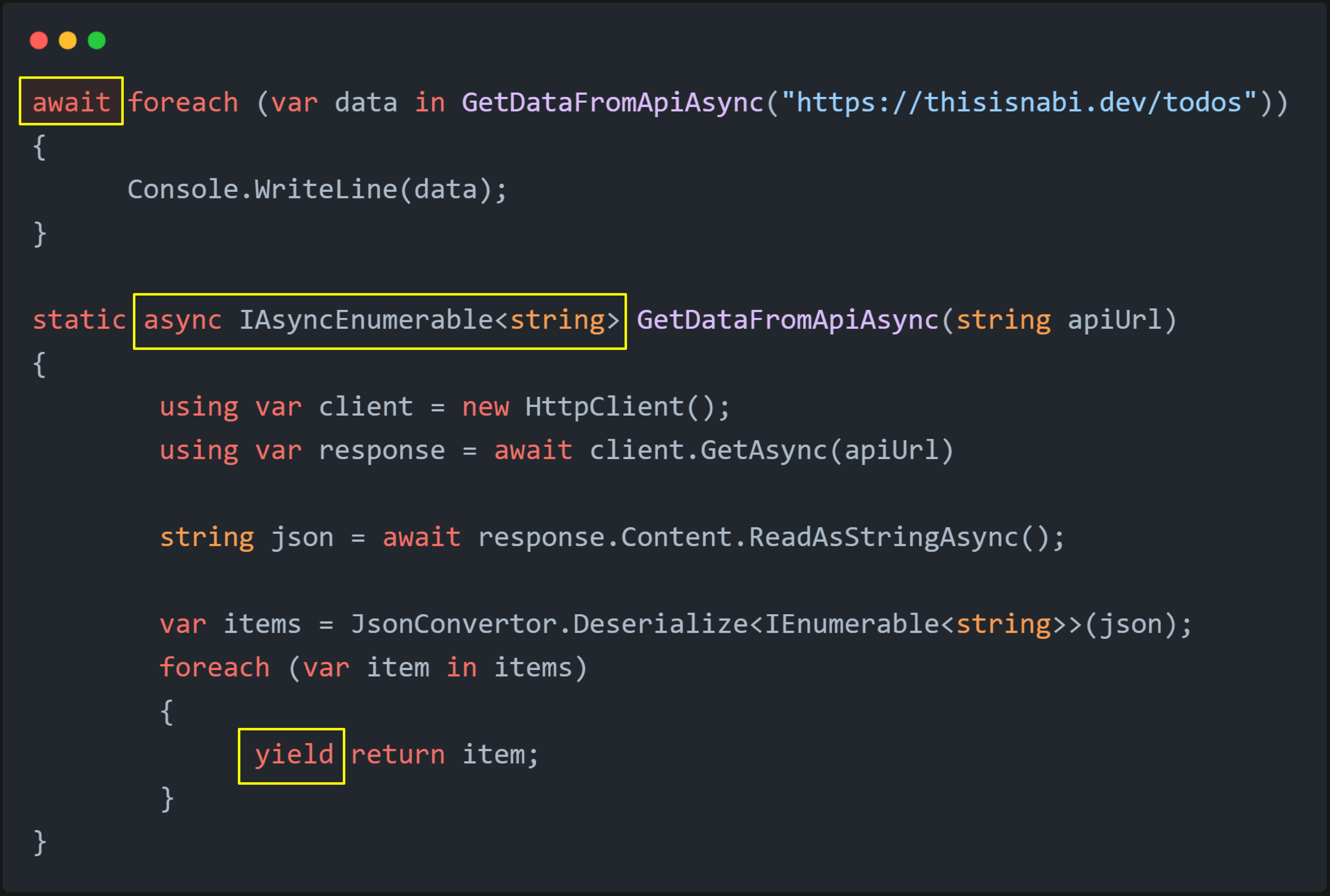
```
string message = null;
Console.WriteLine($"The length of the message is {message.Length}");

string? originalMessage = message;

message = "Hello, World!";
// No warning. Analysis determined "message" is not null.
 Console.WriteLine($"The length of the message is {message.Length}");

Console.WriteLine(originalMessage.Length);
```

⚠ Compiler will warn you when you are potentially using a null reference

Nabi Karampoor
@thisisnabi

6 Asynchronous Streams

Allow asynchronous iteration over a potentially infinite sequence of asynchronous operations



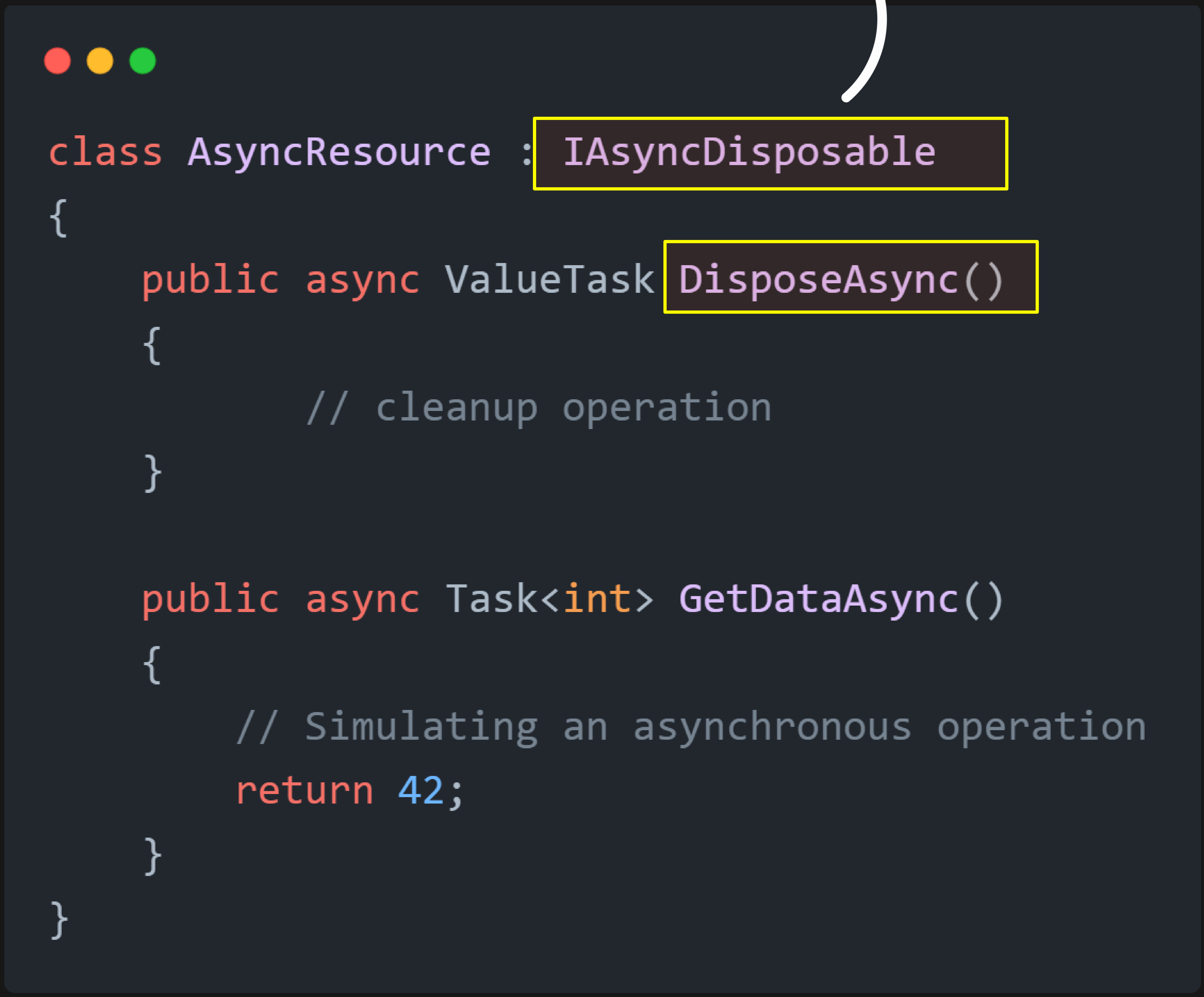
```
await foreach (var data in GetDataFromApiAsync("https://thisisnabi.dev/todos"))
{
      Console.WriteLine(data);
}

static async IAsyncEnumerable<string> GetDataFromApiAsync(string apiUrl)
{
        using var client = new HttpClient();
        using var response = await client.GetAsync(apiUrl)

        string json = await response.Content.ReadAsStringAsync();

        var items = JsonConvertor.Deserialize<IEnumerable<string>>(json);
        foreach (var item in items)
        {
              yield return item;
        }
}
```

Combining async and yield to produce and consume asynchronous data.

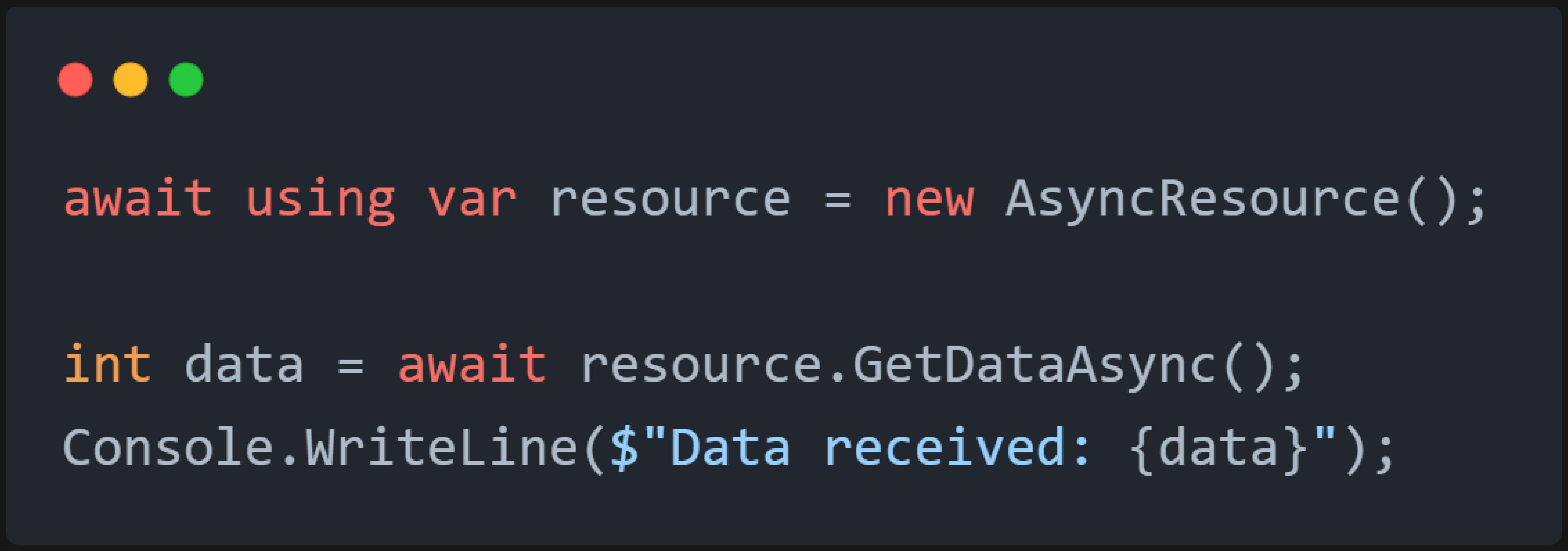
Nabi Karampoor
@thisisnabi

7 Asynchronous Disposable

Allows asynchronous resource cleanup



```
class AsyncResource : IAsyncDisposable
{
    public async ValueTask DisposeAsync()
    {
           // cleanup operation
    }

    public async Task<int> GetDataAsync()
    {
        // Simulating an asynchronous operation
        return 42;
    }
}
```

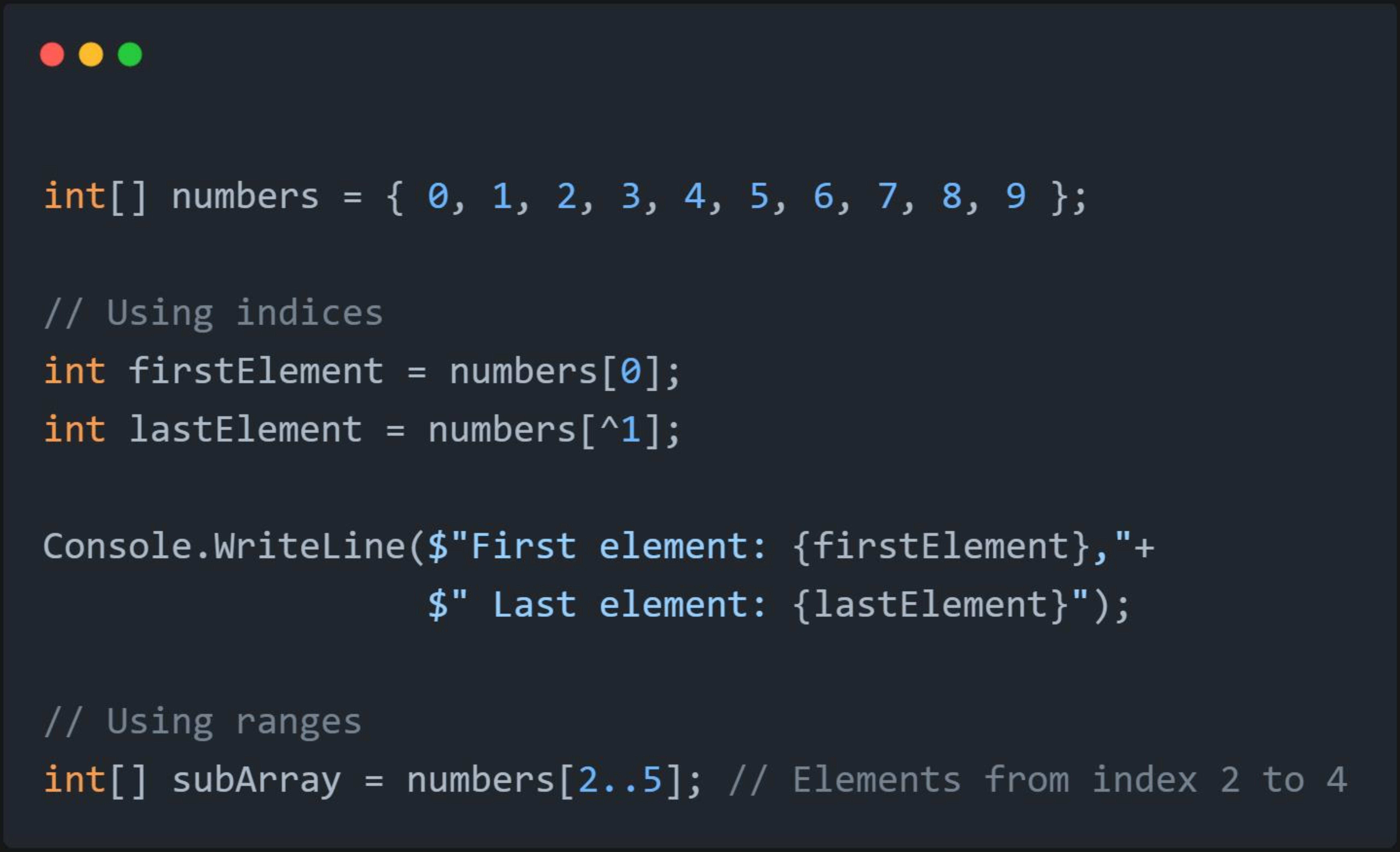


```
await using var resource = new AsyncResource();

int data = await resource.GetDataAsync();
Console.WriteLine($"Data received: {data}");
```

Nabi Karampoor
@thisisnabi

8 Indices and Ranges

Provide concise syntax for working with sequences by allowing you to easily access elements using index and range expressions.



```csharp
int[] numbers = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

// Using indices
int firstElement = numbers[0];
int lastElement = numbers[^1];

Console.WriteLine($"First element: {firstElement},"+
                  $" Last element: {lastElement}");

// Using ranges
int[] subArray = numbers[2..5]; // Elements from index 2 to 4
```

Range A sub-range of the given sequence or collection.

Index An index in the given sequence or collection.

Nabi Karampoor
@thisisnabi