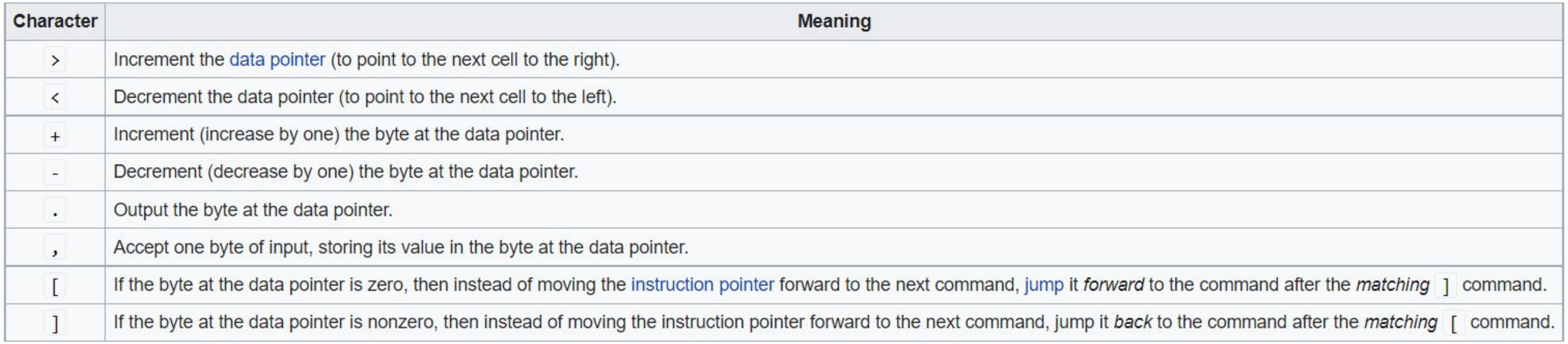
Пиши на C# как тиктокер, а не как дед!

Жмур Анатолий
Broadridge

Минимальные определения алгоритма

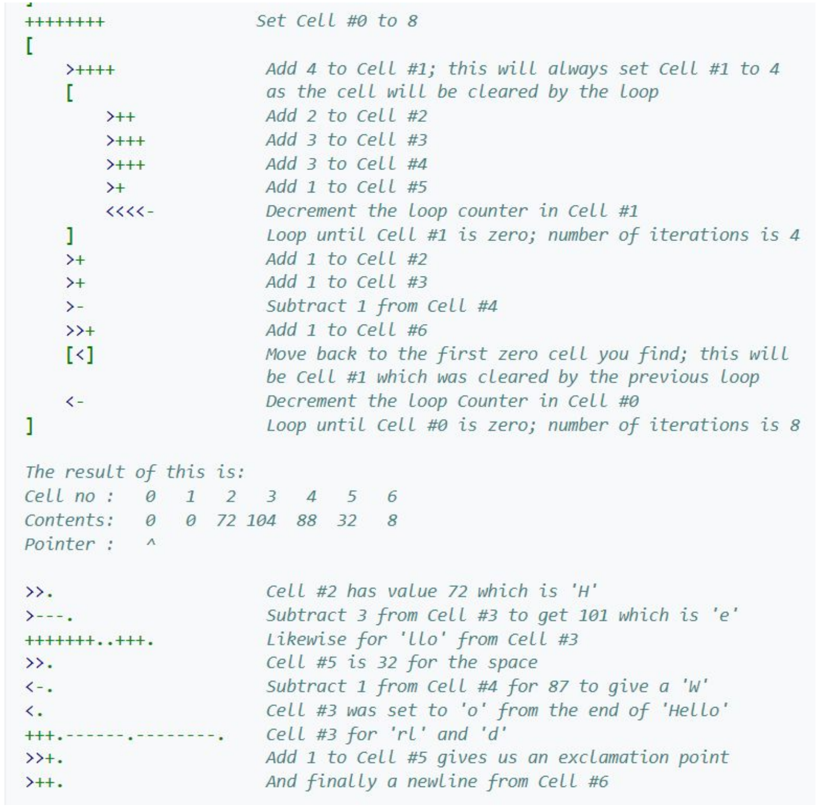
- Машина Тюринга
- Нормальные алгоритмы Маркова
- Эзотерические языки программирования

Brainfuck programming language



Character	Meaning
>	Increment the data pointer (to point to the next cell to the right).
<	Decrement the data pointer (to point to the next cell to the left).
+	Increment (increase by one) the byte at the data pointer.
-	Decrement (decrease by one) the byte at the data pointer.
.	Output the byte at the data pointer.
,	Accept one byte of input, storing its value in the byte at the data pointer.
[	If the byte at the data pointer is zero, then instead of moving the instruction pointer forward to the next command, jump it *forward* to the command after the *matching*] command.
]	If the byte at the data pointer is nonzero, then instead of moving the instruction pointer forward to the next command, jump it *back* to the command after the *matching* [command.

Hello world in Brainfuck

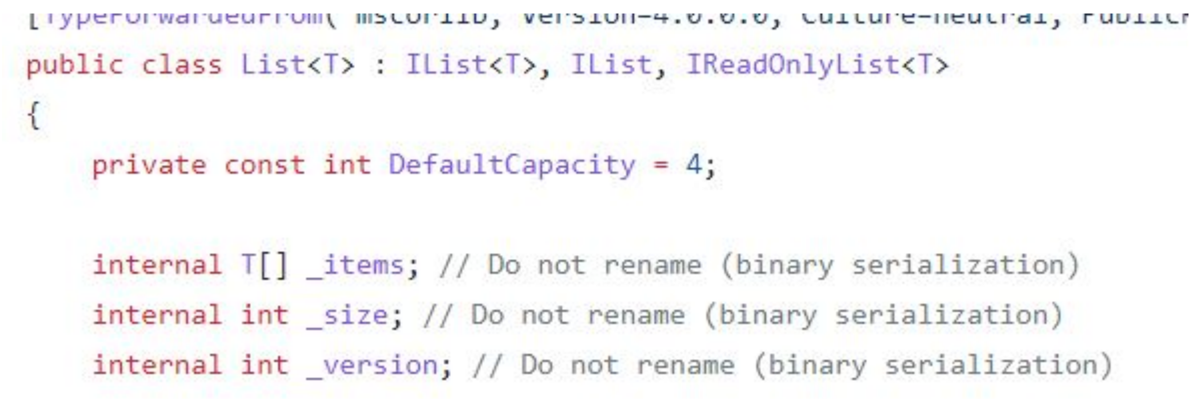


```
++++++++               Set Cell #0 to 8
[
    >++++               Add 4 to Cell #1; this will always set Cell #1 to 4
    [                   as the cell will be cleared by the loop
        >++             Add 2 to Cell #2
        >+++            Add 3 to Cell #3
        >+++            Add 3 to Cell #4
        >+              Add 1 to Cell #5
        <<<<-           Decrement the loop counter in Cell #1
    ]                   Loop until Cell #1 is zero; number of iterations is 4
    >+                  Add 1 to Cell #2
    >+                  Add 1 to Cell #3
    >-                  Subtract 1 from Cell #4
    >>+                 Add 1 to Cell #6
    [<]                 Move back to the first zero cell you find; this will
                        be Cell #1 which was cleared by the previous loop
    <-                  Decrement the loop Counter in Cell #0
]                       Loop until Cell #0 is zero; number of iterations is 8

The result of this is:
Cell no :   0   1   2   3   4   5   6
Contents:   0   0  72 104  88  32   8
Pointer :   ^

>>.                     Cell #2 has value 72 which is 'H'
>---.                   Subtract 3 from Cell #3 to get 101 which is 'e'
+++++++..+++.           Likewise for 'llo' from Cell #3
>>.                     Cell #5 is 32 for the space
<-.                     Subtract 1 from Cell #4 for 87 to give a 'W'
<.                      Cell #3 was set to 'o' from the end of 'Hello'
+++.------.--------.    Cell #3 for 'rl' and 'd'
>>+.                    Add 1 to Cell #5 gives us an exclamation point
>++.                    And finally a newline from Cell #6
```

Современные языки очень избыточны

- В си, например, три цикла и goto (в сишарпе +2 foreach и await foreach)
- Достаточно условного оператор и goto
- Создатели C# решили часть Си грехов типа fallthrough case и приведения к булевскому выражению int и т.п.
- В целом развитие ведет к увелечению избыточности ради удобства в некоторых случаях
- Это называется Syntax Sugar
- https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history

Культура и предубеждения



```
[TypeForwardedFrom("mscorlib, Version=4.0.0.0, Culture=neutral, Public
public class List<T> : IList<T>, IList, IReadOnlyList<T>
{
    private const int DefaultCapacity = 4;

    internal T[] _items; // Do not rename (binary serialization)
    internal int _size; // Do not rename (binary serialization)
    internal int _version; // Do not rename (binary serialization)
```

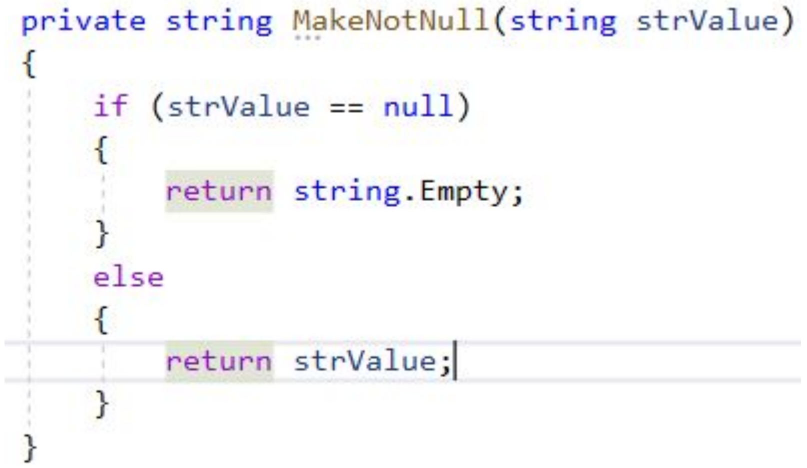
https://github.com/dotnet/runtime/blob/main/src/libraries/System.Private.CoreLib/src/System/Collections/Generic/List.cs

Жми на помидоры! Делай код лучше!

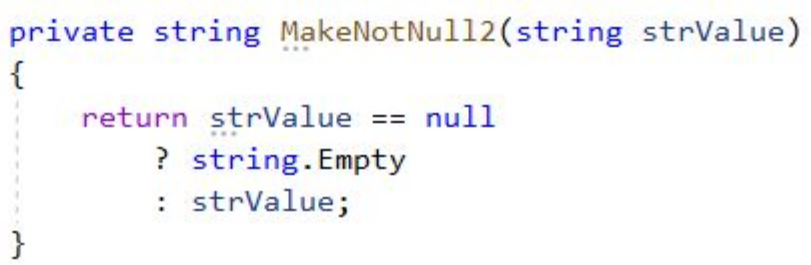
- Современные IDE предлагают много вариантов “улучшений” кода
- К сожалению многие считают себя очень крутыми программистами применяя эти улучшения повсюду
- Это ломает историю!
- Как правило, не стоит “улучшать” код, который вы не меняете по другой причине

File level namespace как исключение

- Не создает существенное изменение для правильно настроенного git
- Уменьшает вложенность

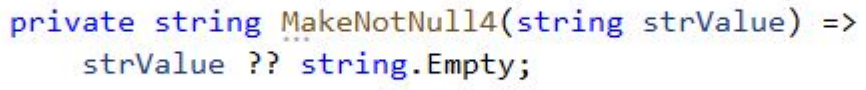


```
private string MakeNotNull(string strValue)
{
    if (strValue == null)
    {
        return string.Empty;
    }
    else
    {
        return strValue;
    }
}
```

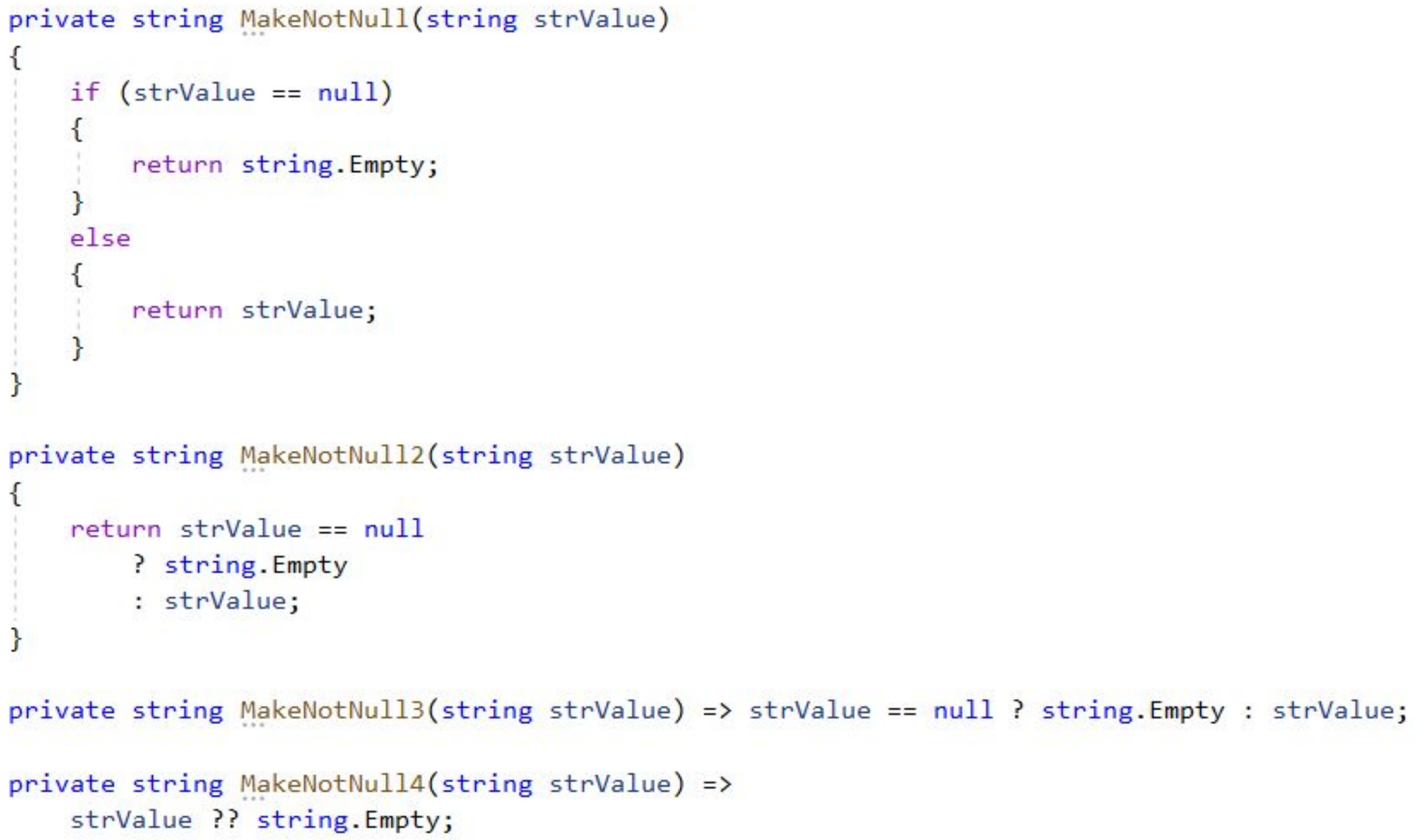


```
private string MakeNotNull2(string strValue)
{
    return strValue == null
        ? string.Empty
        : strValue;
}
```

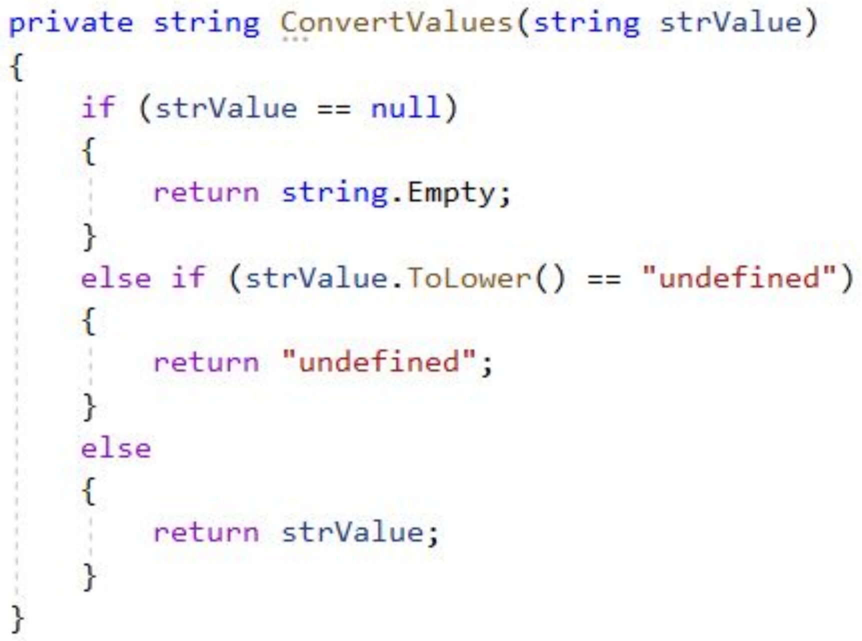
```
private string MakeNotNull3(string strValue) => strValue == null ? string.Empty : strValue;
```



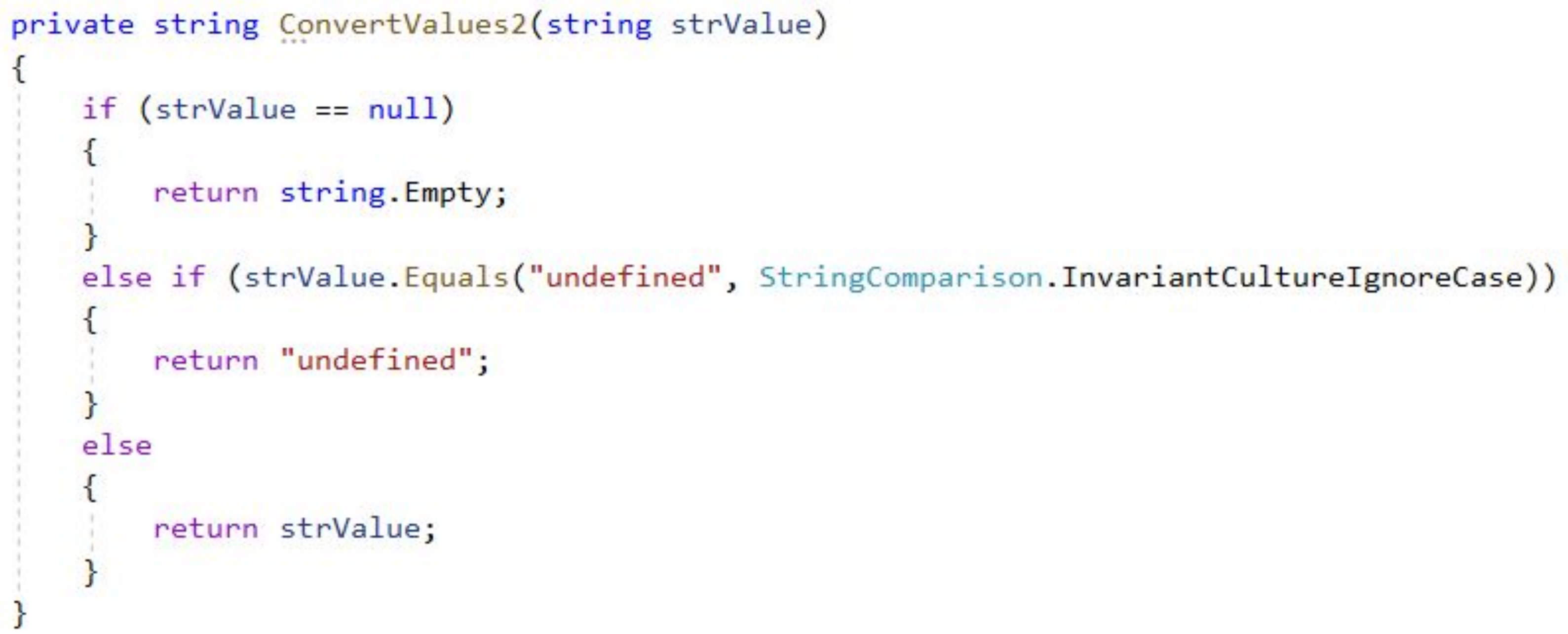
```
private string MakeNotNull4(string strValue) =>
    strValue ?? string.Empty;
```



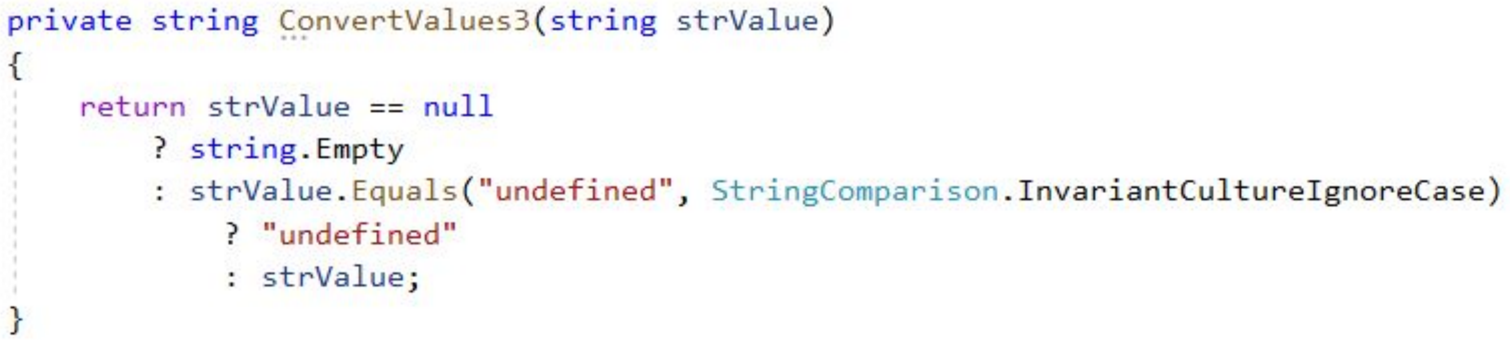
```csharp
private string MakeNotNull(string strValue)
{
    if (strValue == null)
    {
        return string.Empty;
    }
    else
    {
        return strValue;
    }
}

private string MakeNotNull2(string strValue)
{
    return strValue == null
        ? string.Empty
        : strValue;
}

private string MakeNotNull3(string strValue) => strValue == null ? string.Empty : strValue;

private string MakeNotNull4(string strValue) =>
    strValue ?? string.Empty;
```



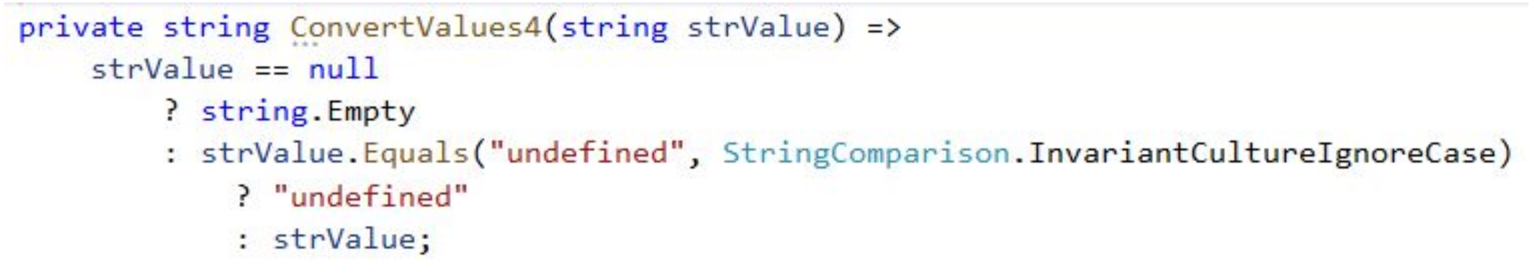
```
private string ConvertValues(string strValue)
{
    if (strValue == null)
    {
        return string.Empty;
    }
    else if (strValue.ToLower() == "undefined")
    {
        return "undefined";
    }
    else
    {
        return strValue;
    }
}
```



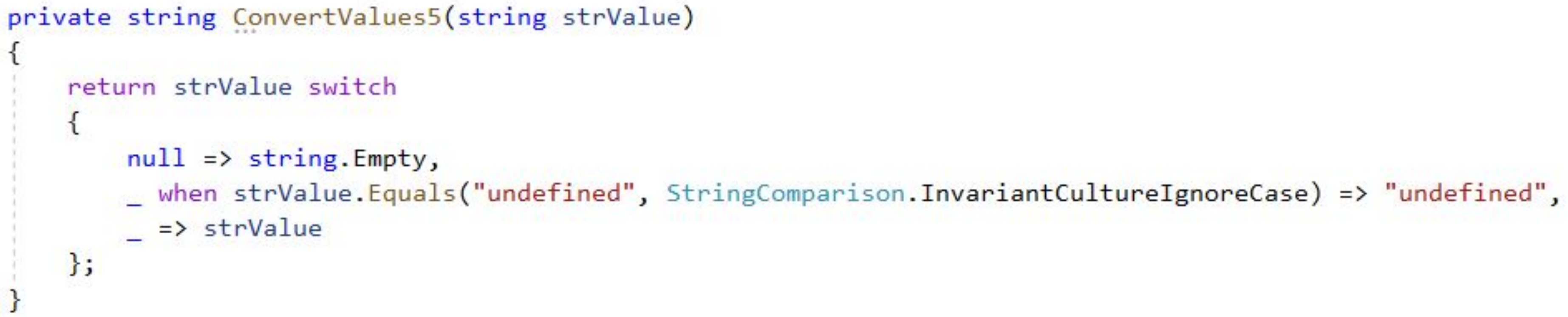
```csharp
private string ConvertValues2(string strValue)
{
    if (strValue == null)
    {
        return string.Empty;
    }
    else if (strValue.Equals("undefined", StringComparison.InvariantCultureIgnoreCase))
    {
        return "undefined";
    }
    else
    {
        return strValue;
    }
}
```



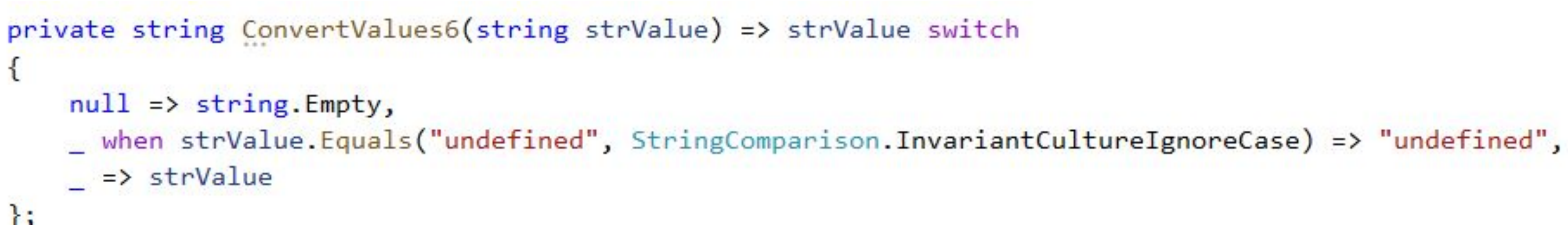
```
private string ConvertValues3(string strValue)
{
    return strValue == null
        ? string.Empty
        : strValue.Equals("undefined", StringComparison.InvariantCultureIgnoreCase)
            ? "undefined"
            : strValue;
}
```



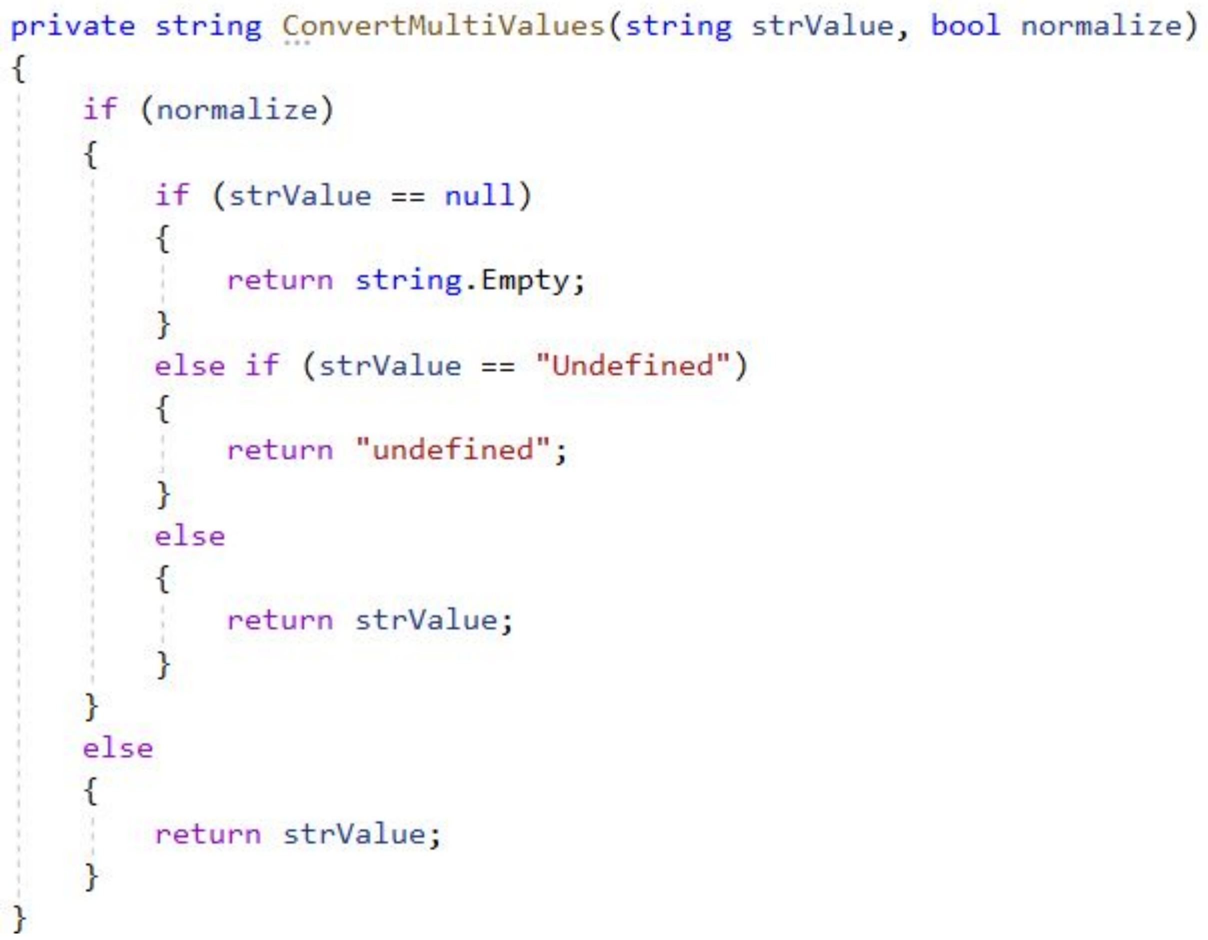
```
private string ConvertValues4(string strValue) =>
    strValue == null
        ? string.Empty
        : strValue.Equals("undefined", StringComparison.InvariantCultureIgnoreCase)
          ? "undefined"
          : strValue;
```



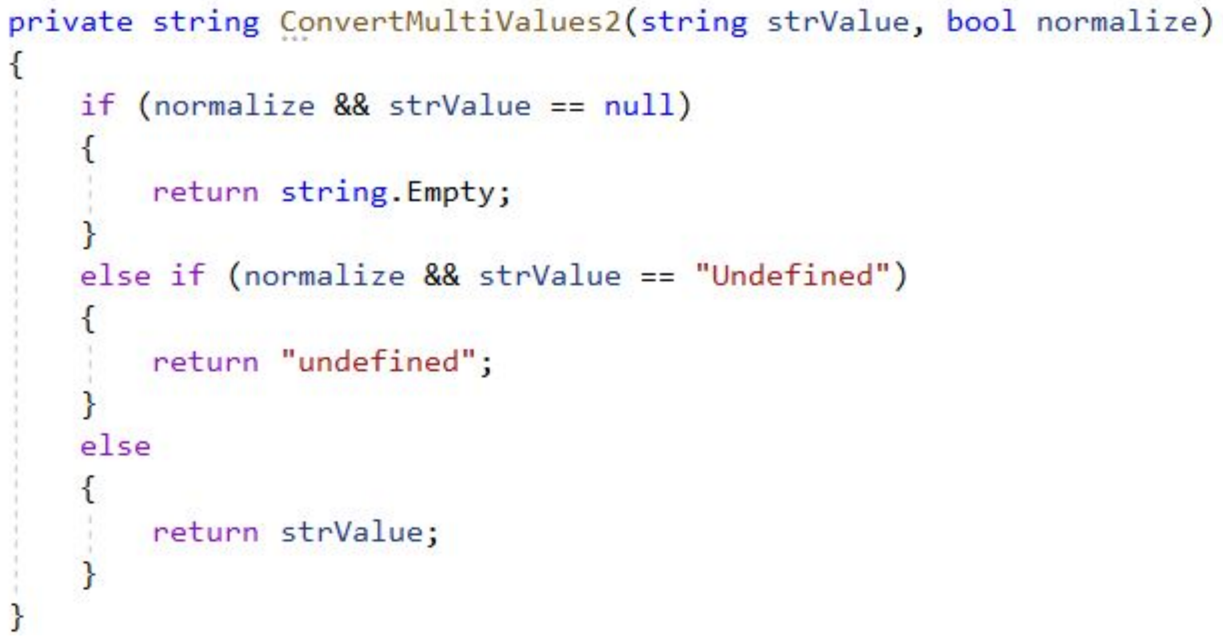
```
private string ConvertValues5(string strValue)
{
    return strValue switch
    {
        null => string.Empty,
        _ when strValue.Equals("undefined", StringComparison.InvariantCultureIgnoreCase) => "undefined",
        _ => strValue
    };
}
```



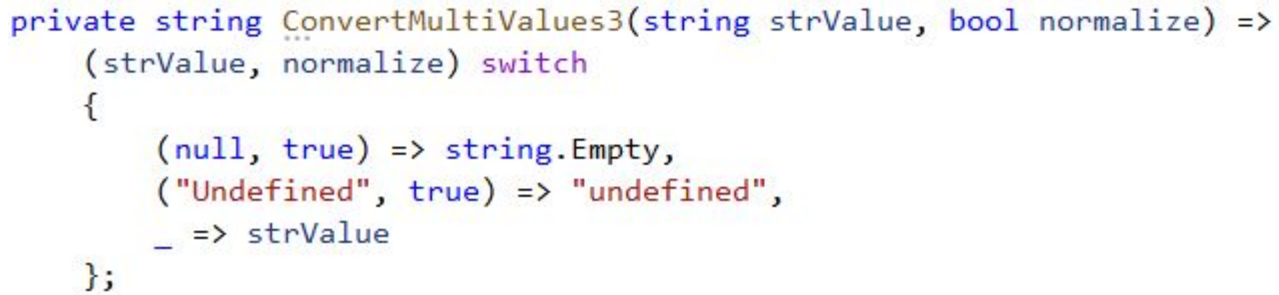
```csharp
private string ConvertValues6(string strValue) => strValue switch
{
    null => string.Empty,
    _ when strValue.Equals("undefined", StringComparison.InvariantCultureIgnoreCase) => "undefined",
    _ => strValue
};
```



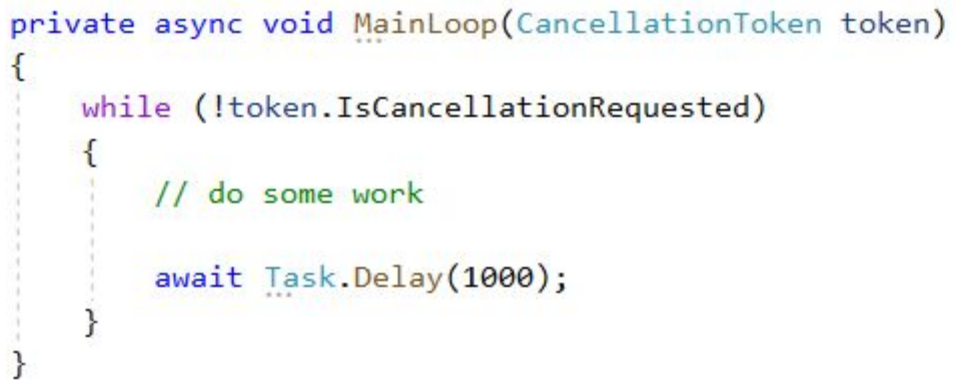
```
private string ConvertMultiValues(string strValue, bool normalize)
{
    if (normalize)
    {
        if (strValue == null)
        {
            return string.Empty;
        }
        else if (strValue == "Undefined")
        {
            return "undefined";
        }
        else
        {
            return strValue;
        }
    }
    else
    {
        return strValue;
    }
}
```



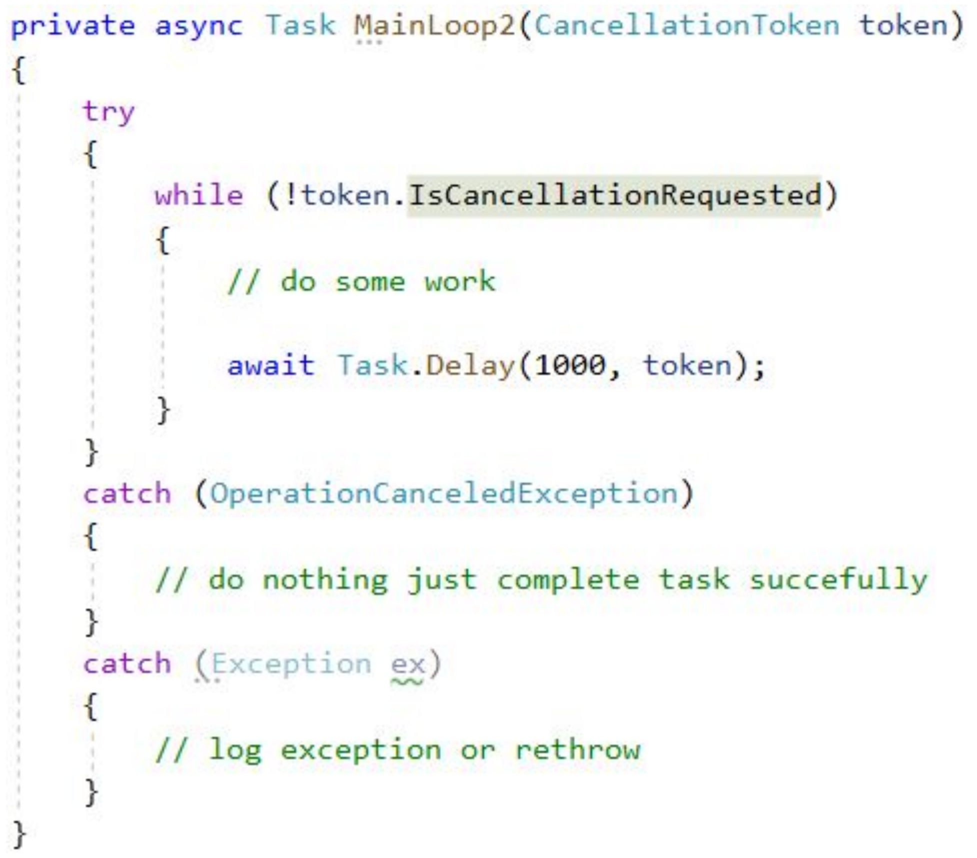
```
private string ConvertMultiValues2(string strValue, bool normalize)
{
    if (normalize && strValue == null)
    {
        return string.Empty;
    }
    else if (normalize && strValue == "Undefined")
    {
        return "undefined";
    }
    else
    {
        return strValue;
    }
}
```



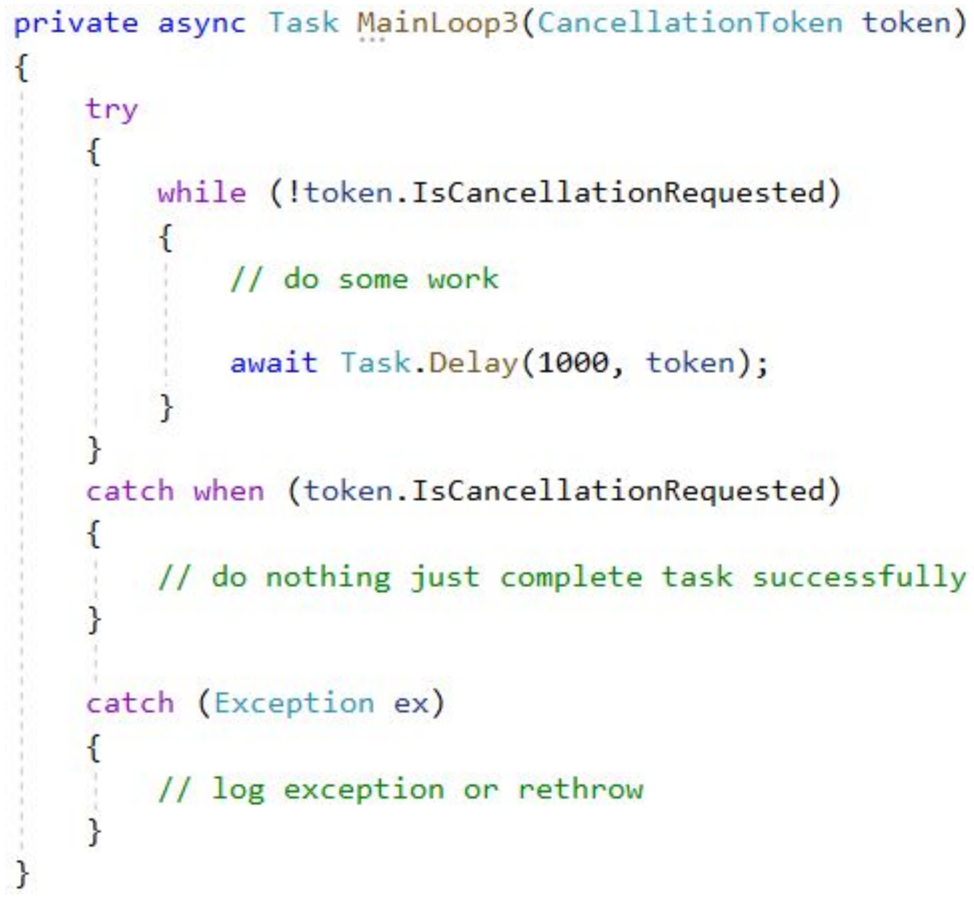
```csharp
private string ConvertMultiValues3(string strValue, bool normalize) =>
    (strValue, normalize) switch
    {
        (null, true) => string.Empty,
        ("Undefined", true) => "undefined",
        _ => strValue
    };
```



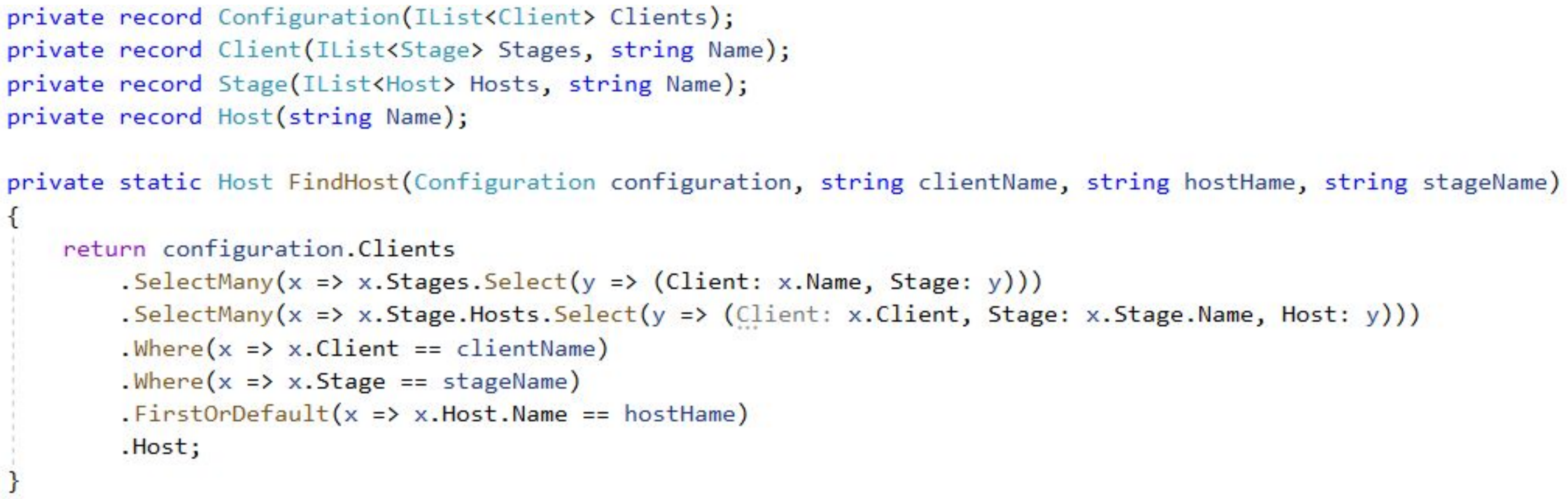
```csharp
private async void MainLoop(CancellationToken token)
{
    while (!token.IsCancellationRequested)
    {
        // do some work

        await Task.Delay(1000);
    }
}
```



```
private async Task MainLoop2(CancellationToken token)
{
    try
    {
        while (!token.IsCancellationRequested)
        {
            // do some work

            await Task.Delay(1000, token);
        }
    }
    catch (OperationCanceledException)
    {
        // do nothing just complete task succefully
    }
    catch (Exception ex)
    {
        // log exception or rethrow
    }
}
```

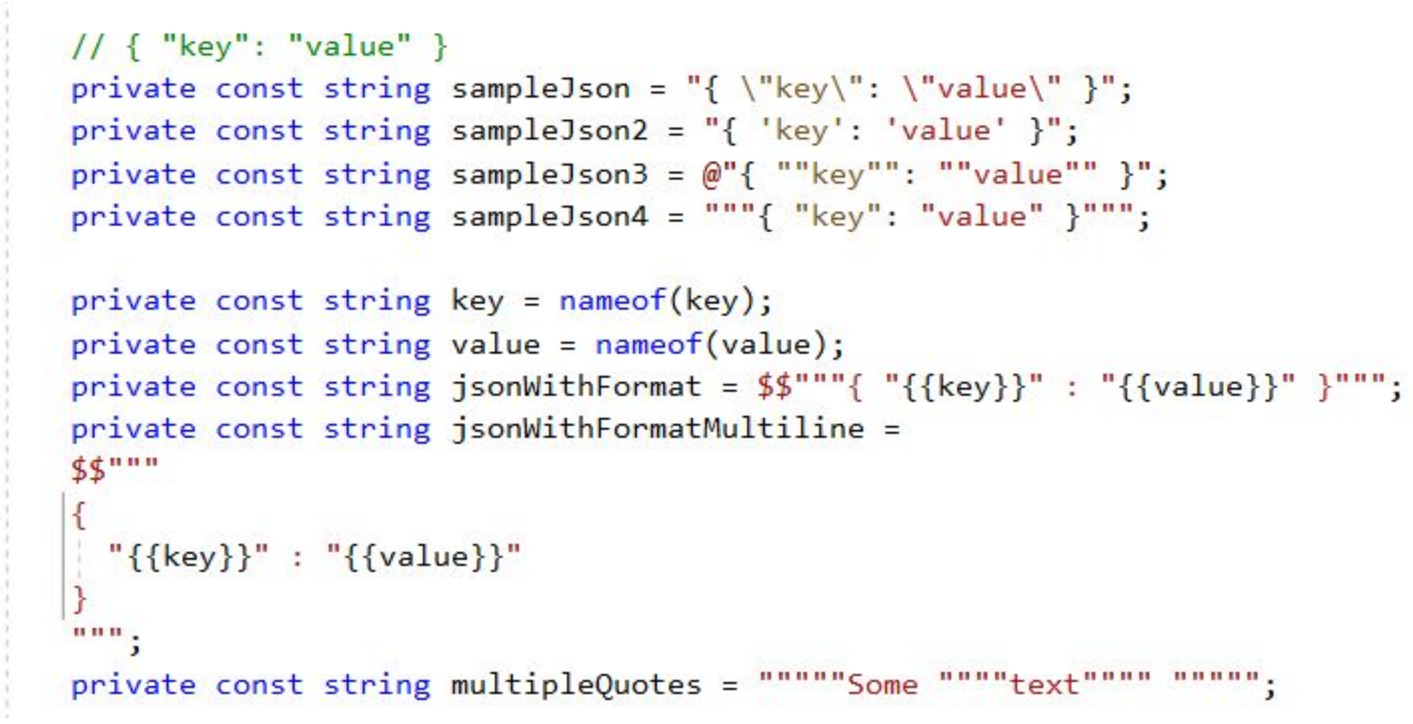


```
private async Task MainLoop3(CancellationToken token)
{
    try
    {
        while (!token.IsCancellationRequested)
        {
            // do some work

            await Task.Delay(1000, token);
        }
    }
    catch when (token.IsCancellationRequested)
    {
        // do nothing just complete task successfully
    }

    catch (Exception ex)
    {
        // log exception or rethrow
    }
}
```

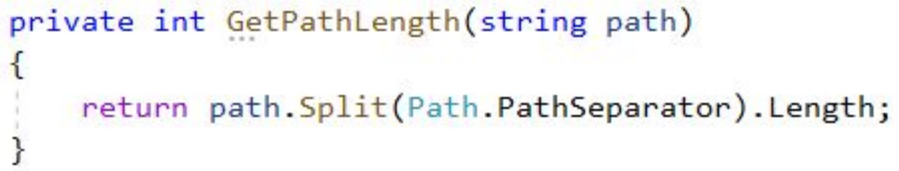


```csharp
private record Configuration(IList<Client> Clients);
private record Client(IList<Stage> Stages, string Name);
private record Stage(IList<Host> Hosts, string Name);
private record Host(string Name);

private static Host FindHost(Configuration configuration, string clientName, string hostHame, string stageName)
{
    return configuration.Clients
        .SelectMany(x => x.Stages.Select(y => (Client: x.Name, Stage: y)))
        .SelectMany(x => x.Stage.Hosts.Select(y => (Client: x.Client, Stage: x.Stage.Name, Host: y)))
        .Where(x => x.Client == clientName)
        .Where(x => x.Stage == stageName)
        .FirstOrDefault(x => x.Host.Name == hostHame)
        .Host;
}
```

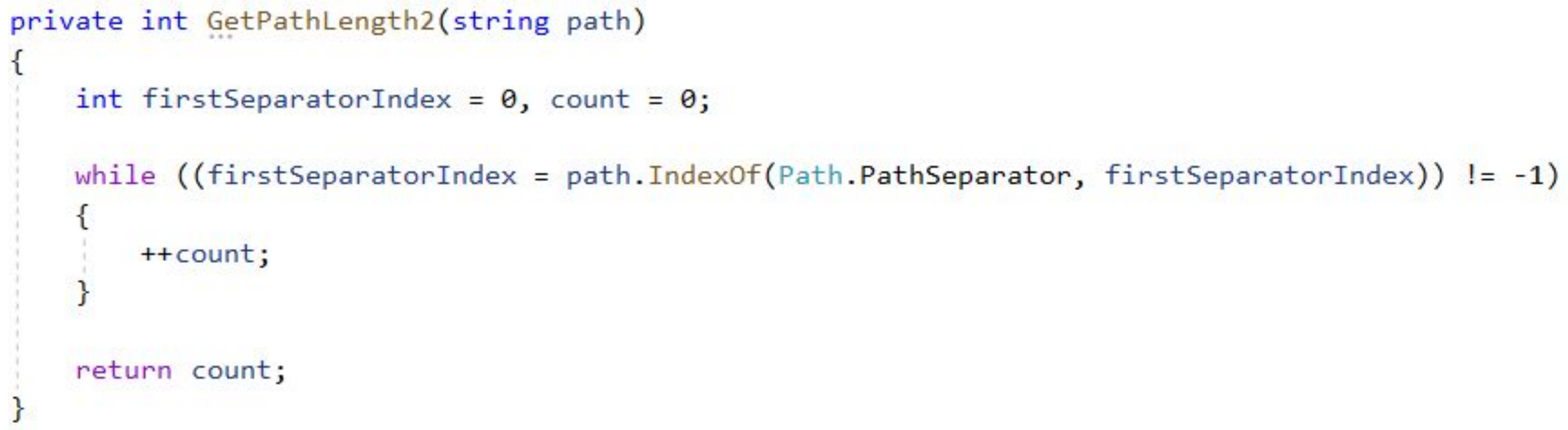
```
private static Host FindHost2(Configuration configuration, string clientName, string hostName, string stageName)
{
    return
        (from client in configuration.Clients
         where client.Name == clientName

         from stage in client.Stages
         where stage.Name == stageName

         from host in stage.Hosts
         where host.Name == hostName

         select host)
         .FirstOrDefault();
}
```



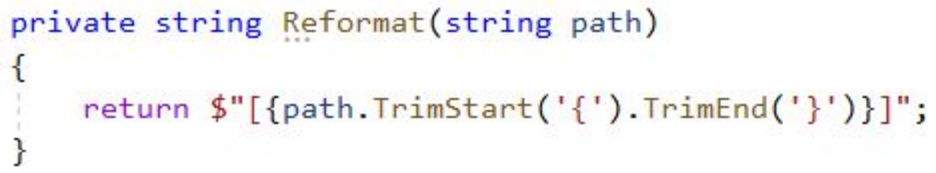
```csharp
// { "key": "value" }
private const string sampleJson = "{ \"key\": \"value\" }";
private const string sampleJson2 = "{ 'key': 'value' }";
private const string sampleJson3 = @"{ ""key"": ""value"" }";
private const string sampleJson4 = """{ "key": "value" }""";

private const string key = nameof(key);
private const string value = nameof(value);
private const string jsonWithFormat = $$"""{ "{{key}}" : "{{value}}" }""";
private const string jsonWithFormatMultiline =
$$"""
{
  "{{key}}" : "{{value}}"
}
""";
private const string multipleQuotes = """""Some """"text"""" """"";
```



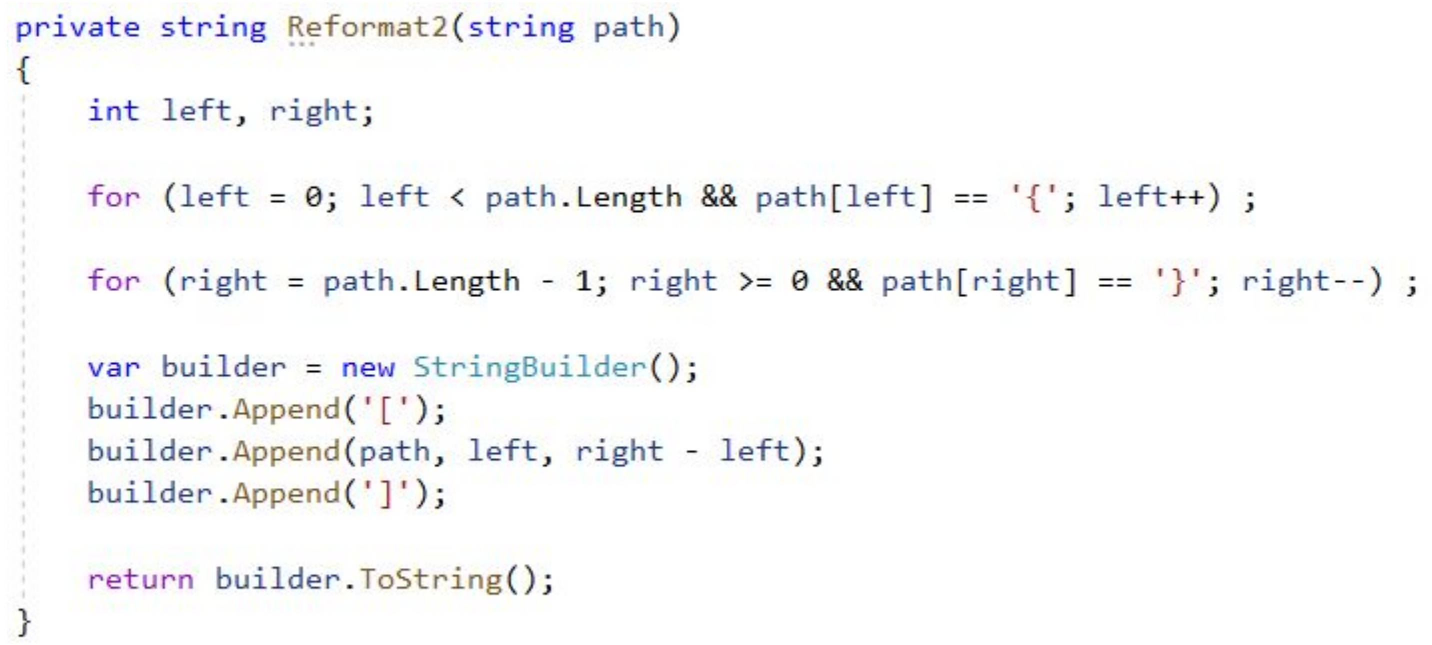
```
private int GetPathLength(string path)
{
    return path.Split(Path.PathSeparator).Length;
}
```



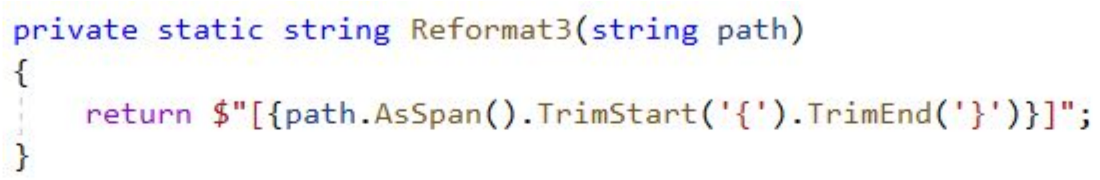
```
private int GetPathLength2(string path)
{
    int firstSeparatorIndex = 0, count = 0;

    while ((firstSeparatorIndex = path.IndexOf(Path.PathSeparator, firstSeparatorIndex)) != -1)
    {
        ++count;
    }

    return count;
}
```



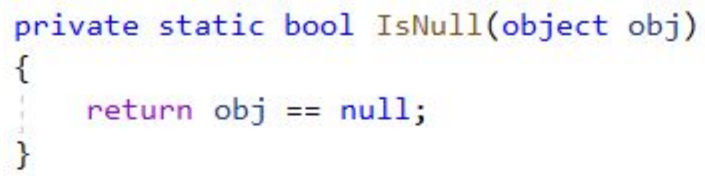
```csharp
private string Reformat(string path)
{
    return $"[{path.TrimStart('{').TrimEnd('}')}]";
}
```



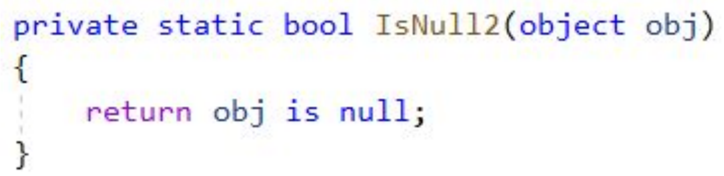
```
private string Reformat2(string path)
{
    int left, right;

    for (left = 0; left < path.Length && path[left] == '{'; left++) ;

    for (right = path.Length - 1; right >= 0 && path[right] == '}'; right--) ;

    var builder = new StringBuilder();
    builder.Append('[');
    builder.Append(path, left, right - left);
    builder.Append(']');

    return builder.ToString();
}
```



```csharp
private static string Reformat3(string path)
{
    return $"[{path.AsSpan().TrimStart('{').TrimEnd('}')}]";
}
```

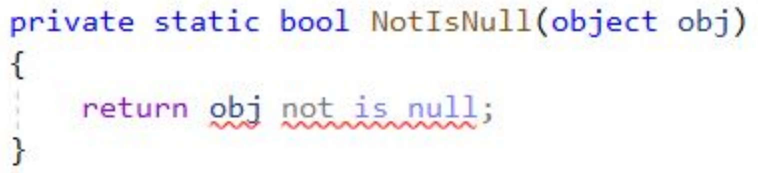


```
private static bool IsNull(object obj)
{
    return obj == null;
}
```

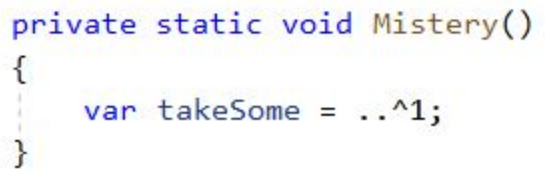


```
private static bool IsNull2(object obj)
{
    return obj is null;
}
```

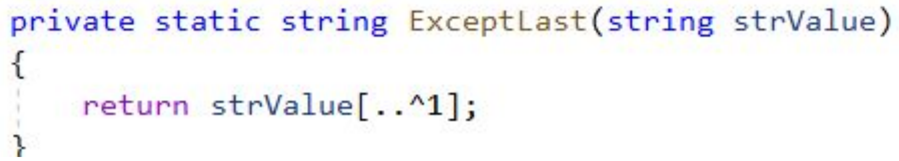
```
private static bool IsNotNull(object obj)
{
    return obj is not null;
}
```



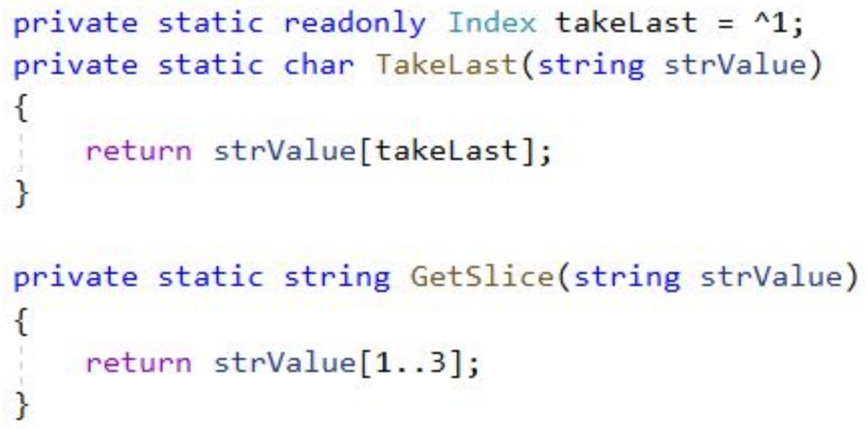
```
private static bool NotIsNull(object obj)
{
    return obj not is null;
}
```



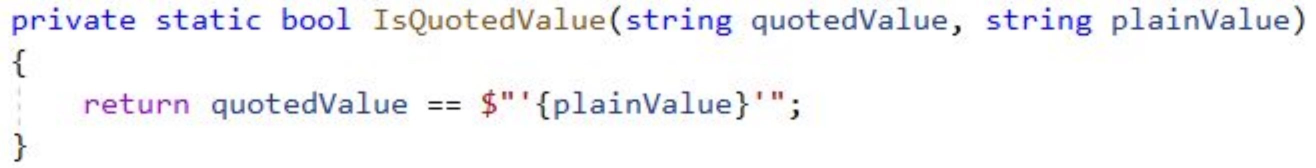
```
private static void Mistery()
{
    var takeSome = ..^1;
}
```



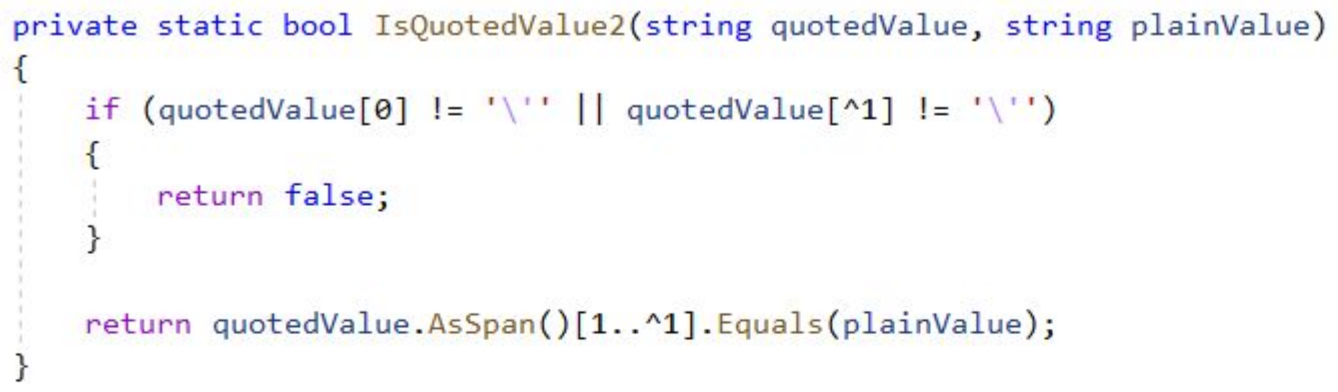
```
private static string ExceptLast(string strValue)
{
    return strValue[..^1];
}
```



```
private static readonly Index takeLast = ^1;
private static char TakeLast(string strValue)
{
    return strValue[takeLast];
}

private static string GetSlice(string strValue)
{
    return strValue[1..3];
}
```

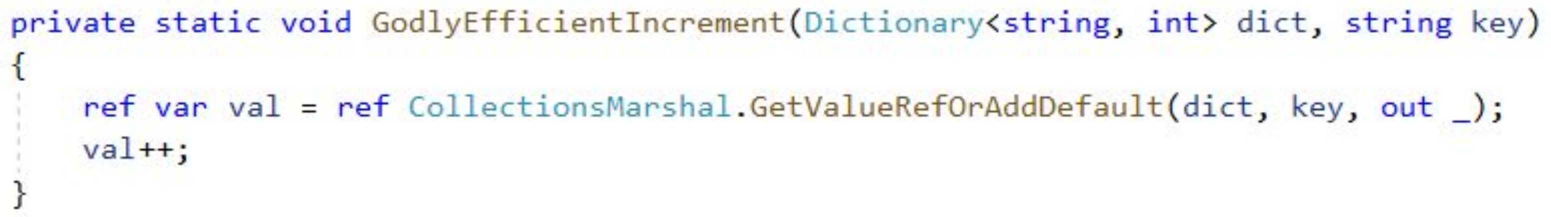


```csharp
private static bool IsQuotedValue(string quotedValue, string plainValue)
{
    return quotedValue == $"'{plainValue}'";
}
```



```csharp
private static bool IsQuotedValue2(string quotedValue, string plainValue)
{
    if (quotedValue[0] != '\'' || quotedValue[^1] != '\'')
    {
        return false;
    }

    return quotedValue.AsSpan()[1..^1].Equals(plainValue);
}
```

```
private static void SimpleIncrement(Dictionary<string, int> dict, string key)
{
    dict[key]++;
}
```

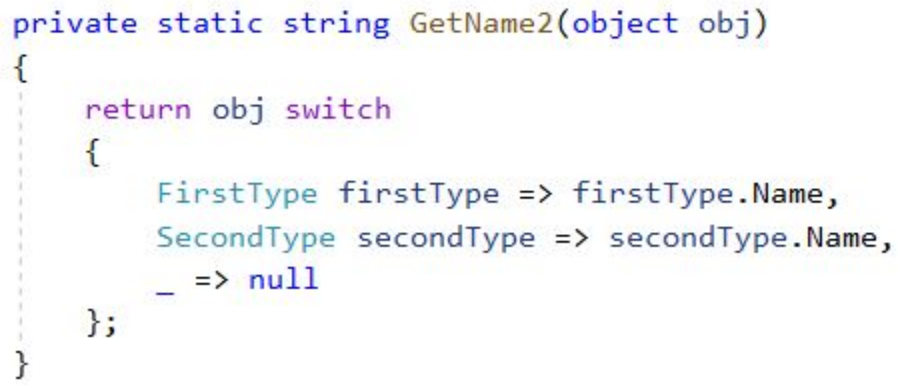


```
private static void GodlyEfficientIncrement(Dictionary<string, int> dict, string key)
{
    ref var val = ref CollectionsMarshal.GetValueRefOrAddDefault(dict, key, out _);
    val++;
}
```

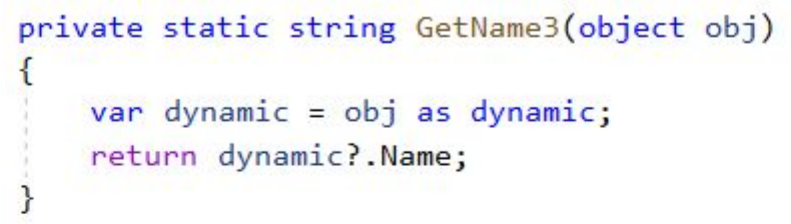
```csharp
private static void IncrementAll(int[] data)
{
    foreach (int item in data)
    {
        item++;
    }
}
```

```
private static void IncrementAll2(int[] data)
{
    foreach (ref int item in data.AsSpan())
    {
        item++;
    }
}
```

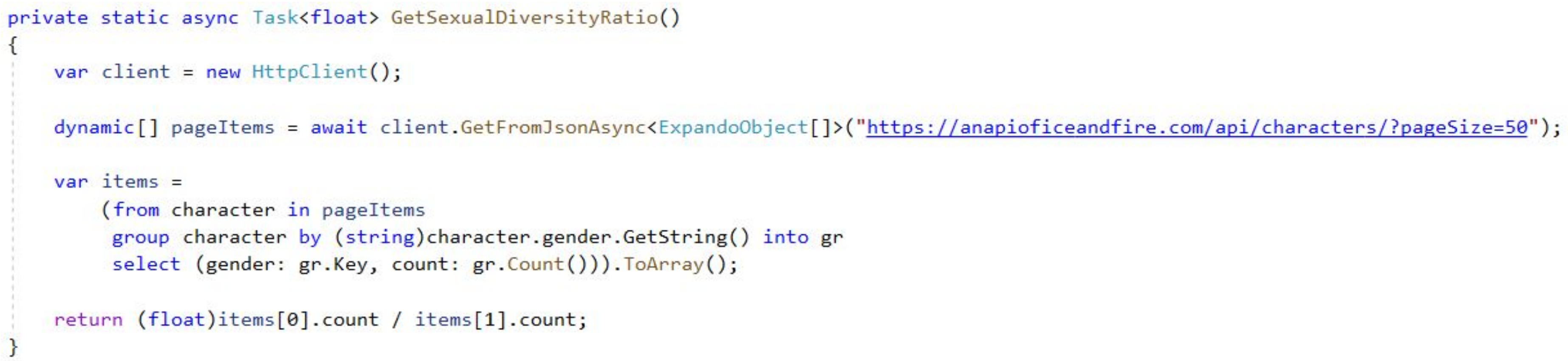
```csharp
private record FirstType(string Name);
private record SecondType(string Name);

private static string GetName(object obj)
{
    if (obj is FirstType firstType)
    {
        return firstType.Name;
    }
    else if (obj is SecondType secondType)
    {
        return secondType.Name;
    }

    return null;
}
```



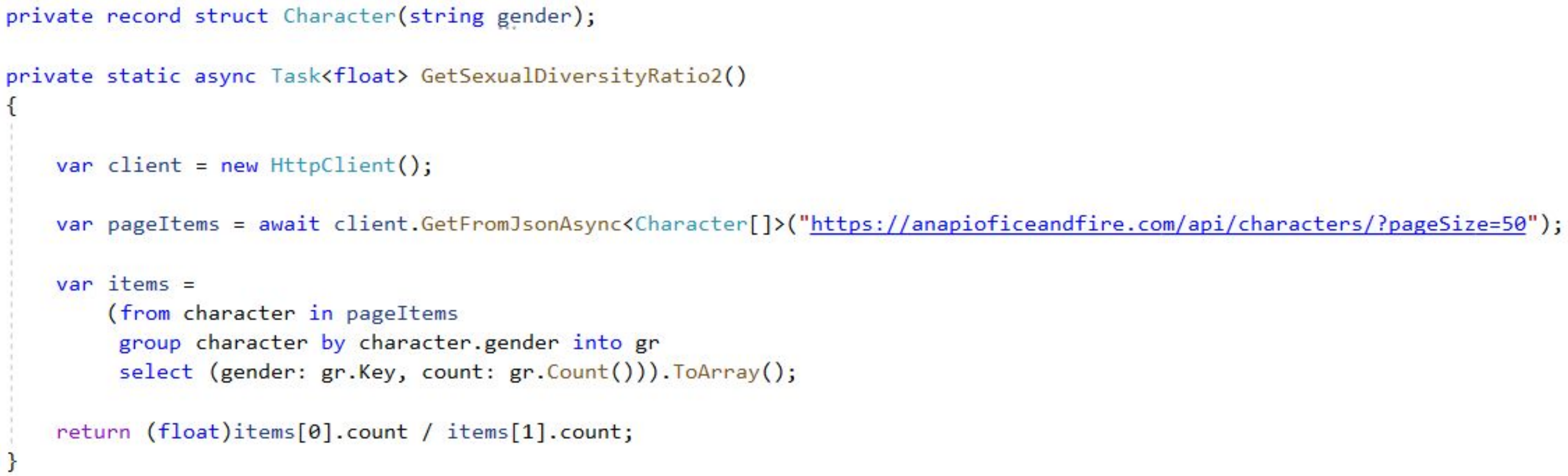
```
private static string GetName2(object obj)
{
    return obj switch
    {
        FirstType firstType => firstType.Name,
        SecondType secondType => secondType.Name,
        _ => null
    };
}
```



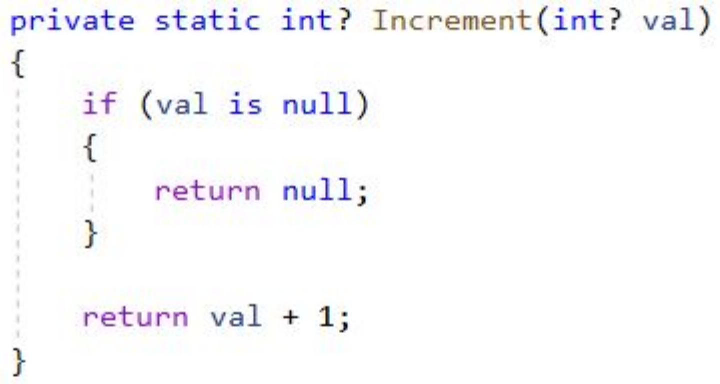
```
private static string GetName3(object obj)
{
    var dynamic = obj as dynamic;
    return dynamic?.Name;
}
```



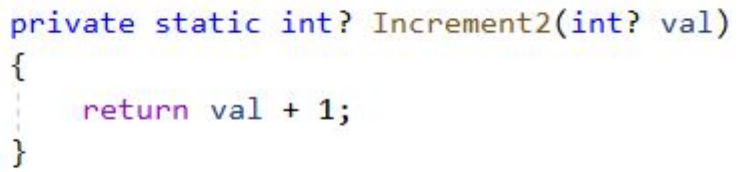
```csharp
private static async Task<float> GetSexualDiversityRatio()
{
    var client = new HttpClient();

    dynamic[] pageItems = await client.GetFromJsonAsync<ExpandoObject[]>("https://anapioficeandfire.com/api/characters/?pageSize=50");

    var items =
        (from character in pageItems
         group character by (string)character.gender.GetString() into gr
         select (gender: gr.Key, count: gr.Count())).ToArray();

    return (float)items[0].count / items[1].count;
}
```



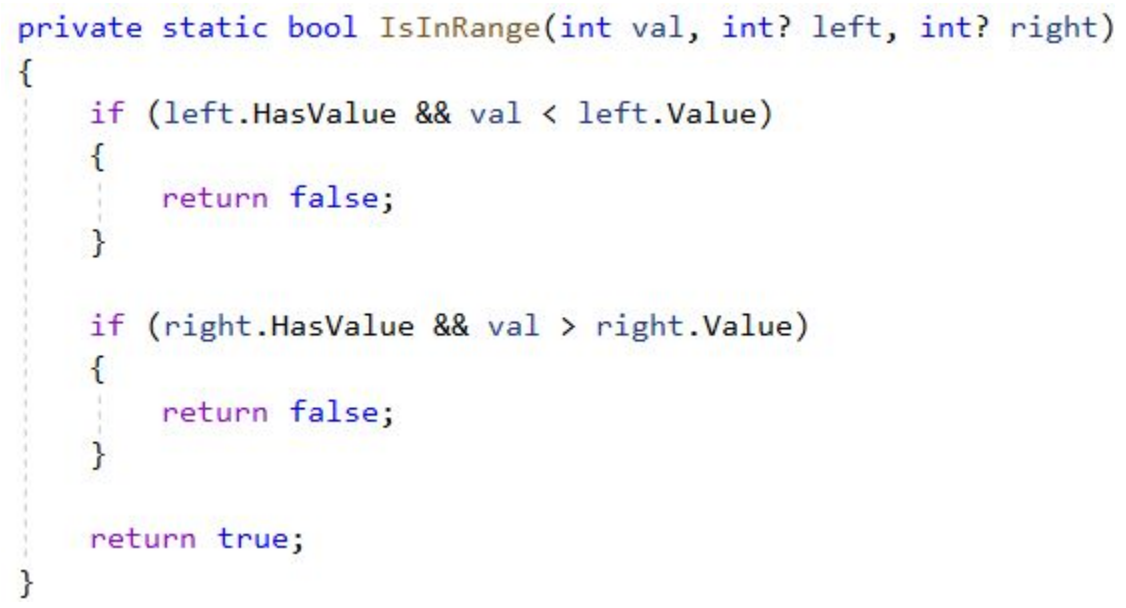
```csharp
private record struct Character(string gender);

private static async Task<float> GetSexualDiversityRatio2()
{

    var client = new HttpClient();

    var pageItems = await client.GetFromJsonAsync<Character[]>("https://anapioficeandfire.com/api/characters/?pageSize=50");

    var items =
        (from character in pageItems
         group character by character.gender into gr
         select (gender: gr.Key, count: gr.Count())).ToArray();

    return (float)items[0].count / items[1].count;
}
```



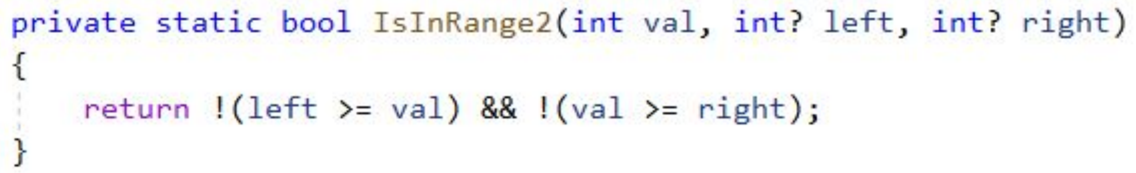
```csharp
private static int? Increment(int? val)
{
    if (val is null)
    {
        return null;
    }

    return val + 1;
}
```



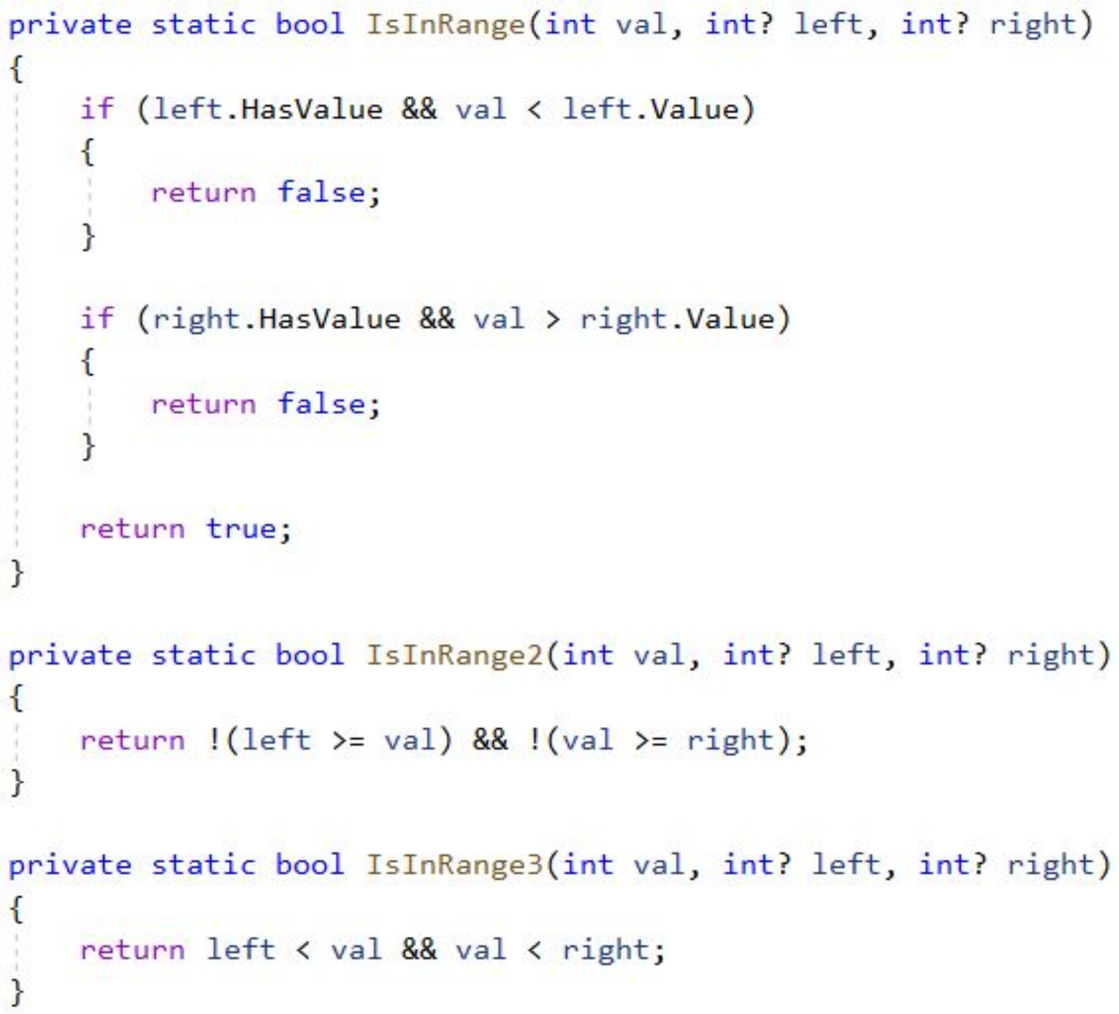
```
private static int? Increment2(int? val)
{
    return val + 1;
}
```



```
private static bool IsInRange(int val, int? left, int? right)
{
    if (left.HasValue && val < left.Value)
    {
        return false;
    }

    if (right.HasValue && val > right.Value)
    {
        return false;
    }

    return true;
}
```

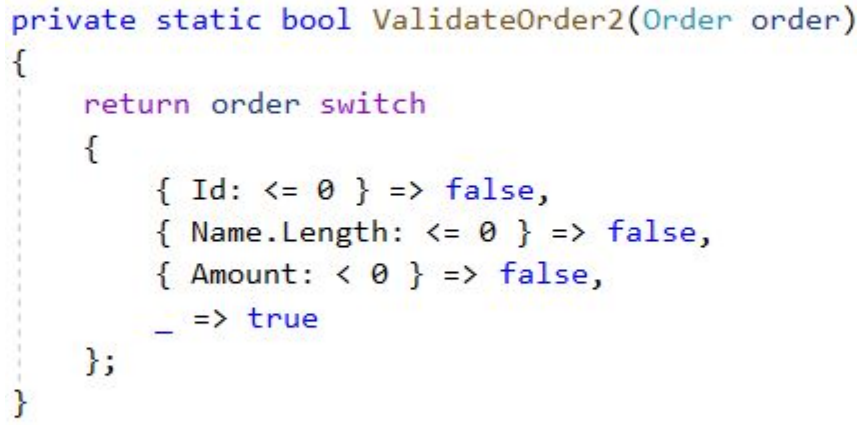


```
private static bool IsInRange2(int val, int? left, int? right)
{
    return !(left >= val) && !(val >= right);
}
```



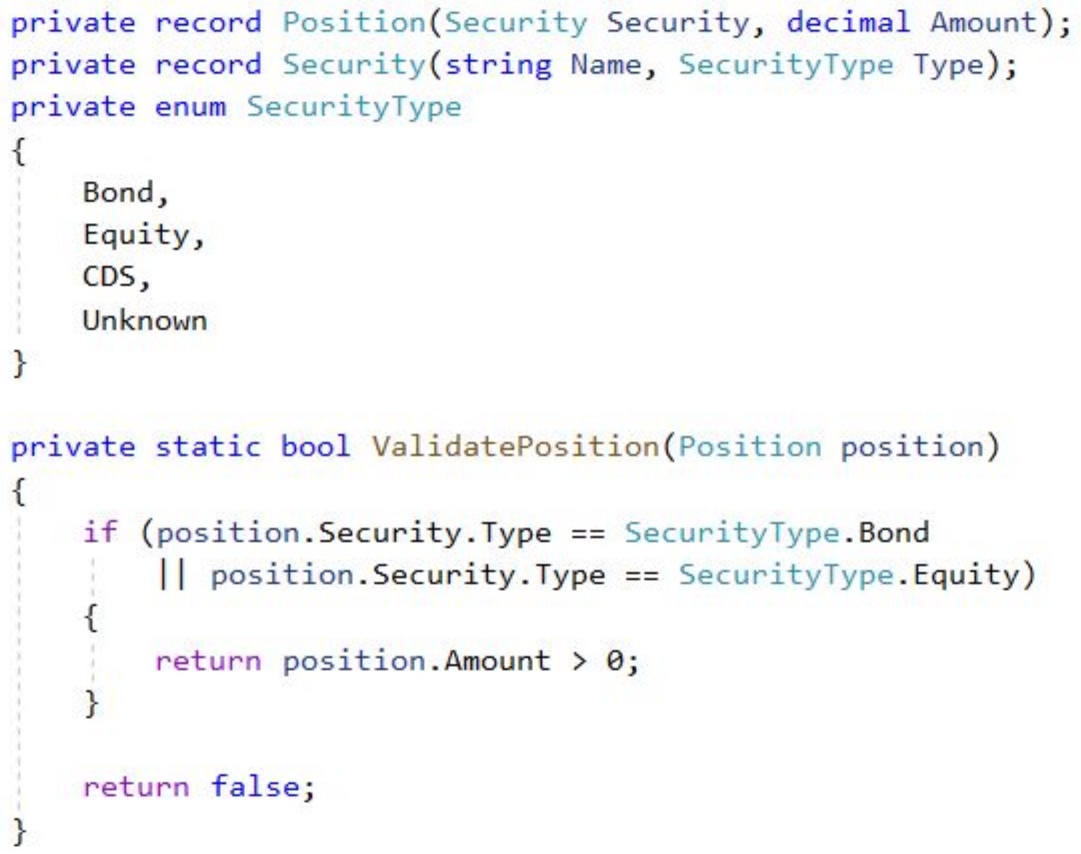
```csharp
private static bool IsInRange(int val, int? left, int? right)
{
    if (left.HasValue && val < left.Value)
    {
        return false;
    }

    if (right.HasValue && val > right.Value)
    {
        return false;
    }

    return true;
}

private static bool IsInRange2(int val, int? left, int? right)
{
    return !(left >= val) && !(val >= right);
}

private static bool IsInRange3(int val, int? left, int? right)
{
    return left < val && val < right;
}
```

```
private record Order(int Id, string Name, double Amount);

private static bool ValidateOrder(Order order)
{
    return order.Id < 0
        || order.Name.Length <= 0
        || order.Amount < 0;
}
```

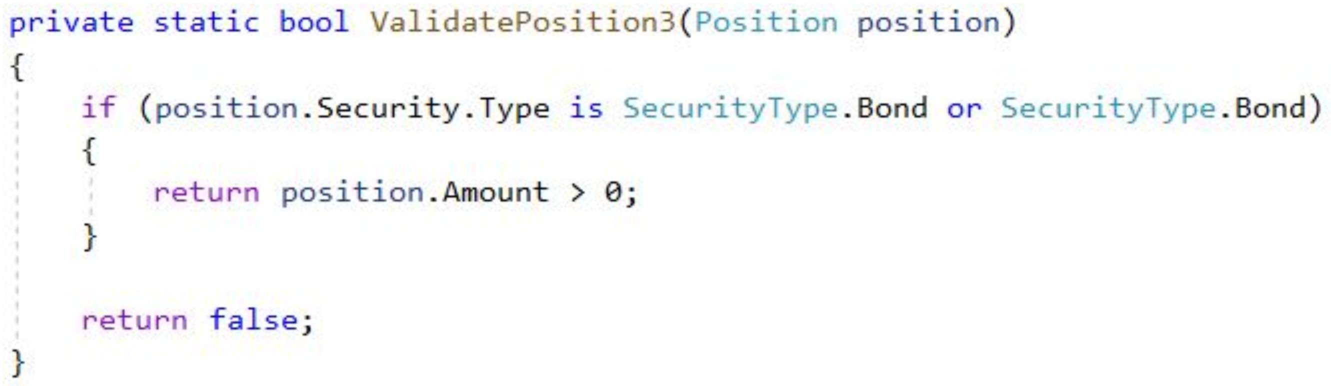


```
private static bool ValidateOrder2(Order order)
{
    return order switch
    {
        { Id: <= 0 } => false,
        { Name.Length: <= 0 } => false,
        { Amount: < 0 } => false,
        _ => true
    };
}
```

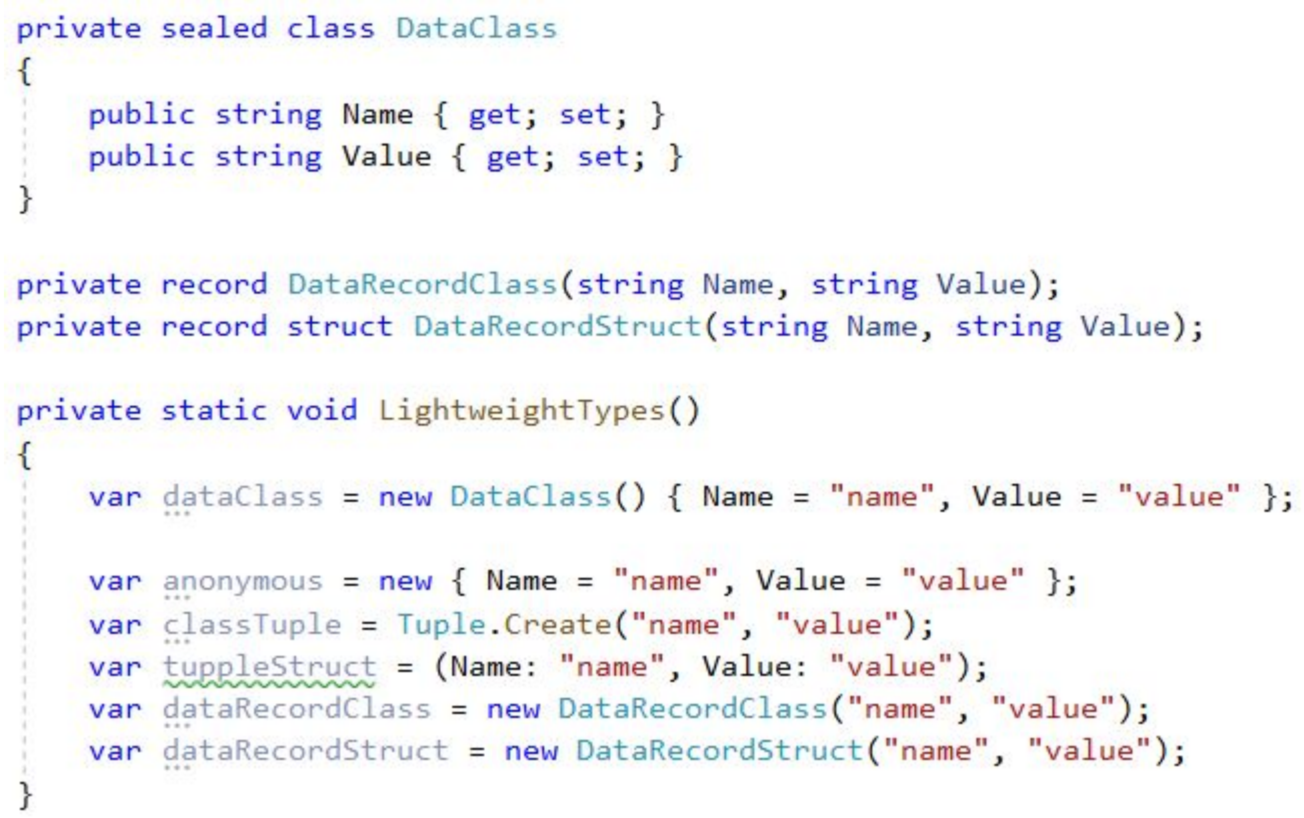
```
private static bool ValidateOrder3(Order order)
{
    return order is not ({ Id: <= 0 } or { Name.Length: <= 0 } or { Amount: < 0 });
}
```



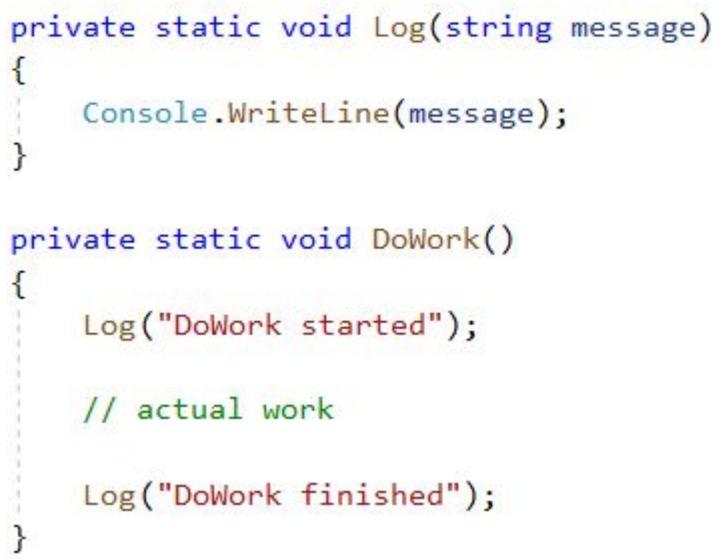
```csharp
private record Position(Security Security, decimal Amount);
private record Security(string Name, SecurityType Type);
private enum SecurityType
{
    Bond,
    Equity,
    CDS,
    Unknown
}

private static bool ValidatePosition(Position position)
{
    if (position.Security.Type == SecurityType.Bond
        || position.Security.Type == SecurityType.Equity)
    {
        return position.Amount > 0;
    }

    return false;
}
```



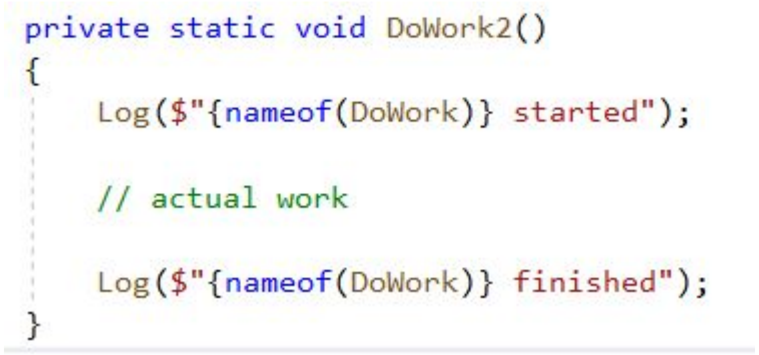
```
private static bool ValidatePosition3(Position position)
{
    if (position.Security.Type is SecurityType.Bond or SecurityType.Bond)
    {
        return position.Amount > 0;
    }

    return false;
}
```



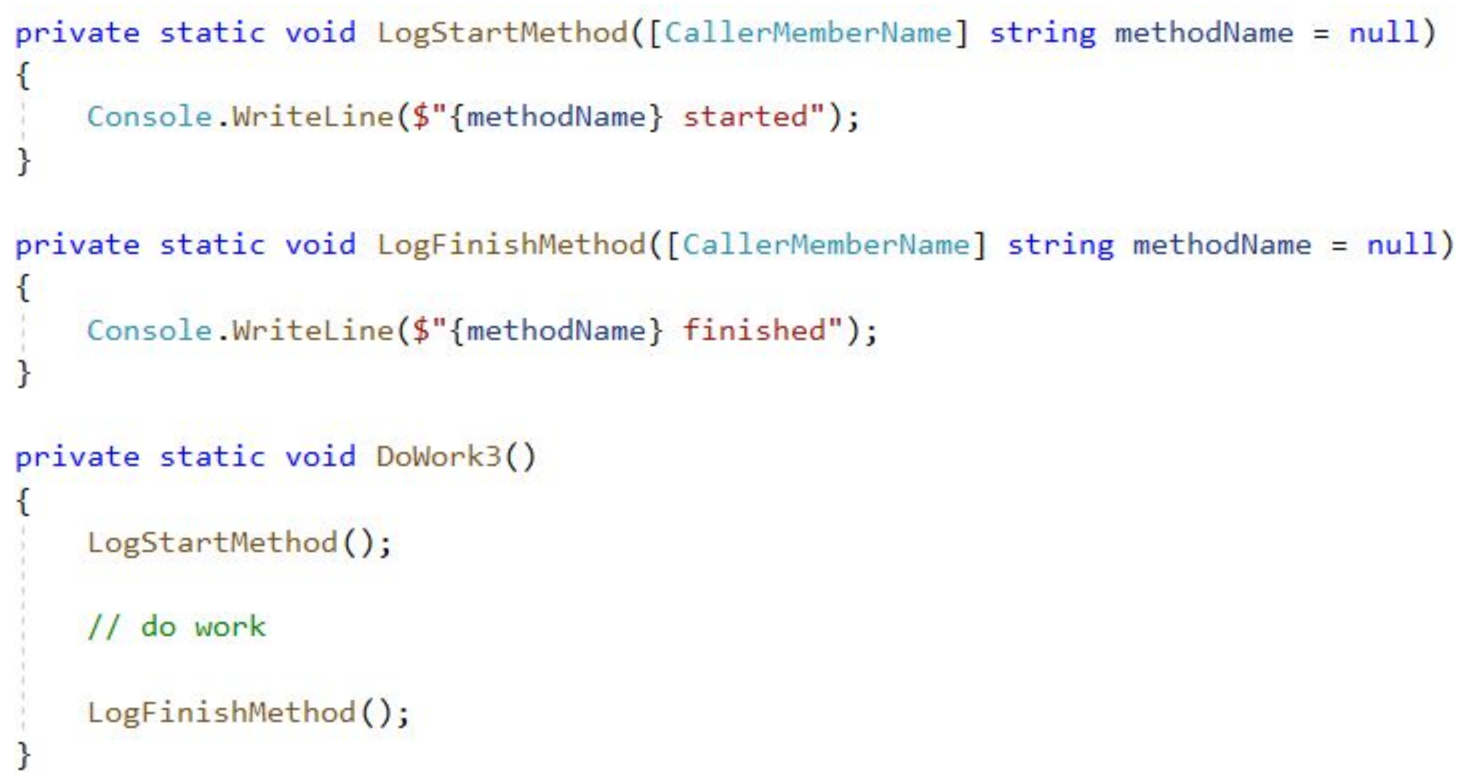
```csharp
private sealed class DataClass
{
    public string Name { get; set; }
    public string Value { get; set; }
}

private record DataRecordClass(string Name, string Value);
private record struct DataRecordStruct(string Name, string Value);

private static void LightweightTypes()
{
    var dataClass = new DataClass() { Name = "name", Value = "value" };

    var anonymous = new { Name = "name", Value = "value" };
    var classTuple = Tuple.Create("name", "value");
    var tuppleStruct = (Name: "name", Value: "value");
    var dataRecordClass = new DataRecordClass("name", "value");
    var dataRecordStruct = new DataRecordStruct("name", "value");
}
```



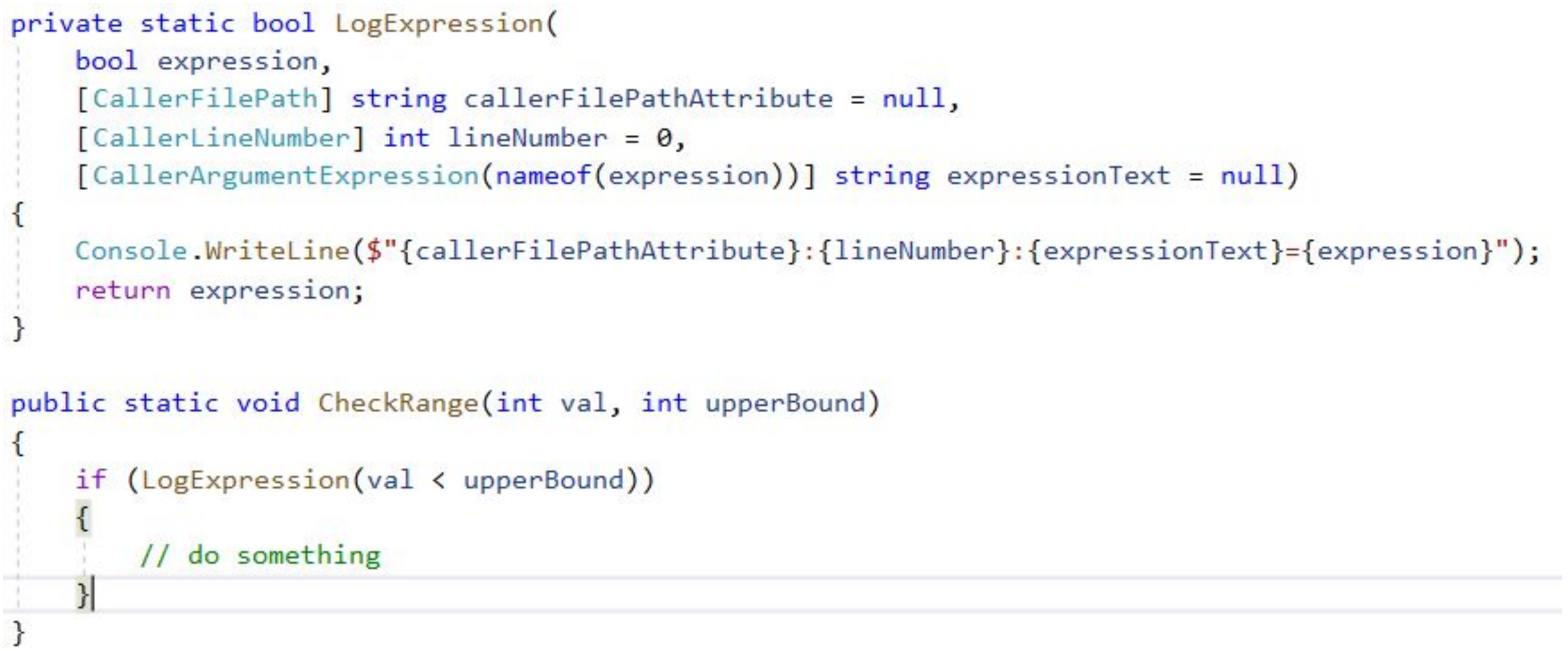
```
private static void Log(string message)
{
    Console.WriteLine(message);
}

private static void DoWork()
{
    Log("DoWork started");

    // actual work

    Log("DoWork finished");
}
```



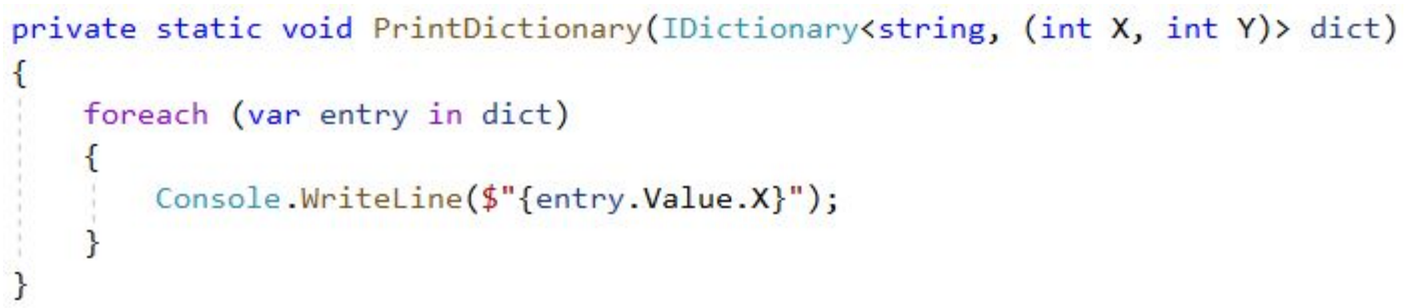
```
private static void DoWork2()
{
    Log($"{nameof(DoWork)} started");

    // actual work

    Log($"{nameof(DoWork)} finished");
}
```



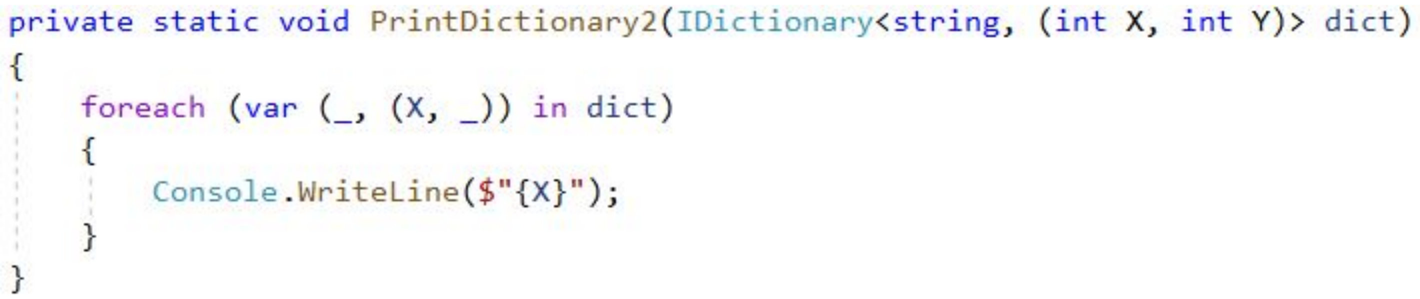
```csharp
private static void LogStartMethod([CallerMemberName] string methodName = null)
{
    Console.WriteLine($"{methodName} started");
}

private static void LogFinishMethod([CallerMemberName] string methodName = null)
{
    Console.WriteLine($"{methodName} finished");
}

private static void DoWork3()
{
    LogStartMethod();

    // do work

    LogFinishMethod();
}
```



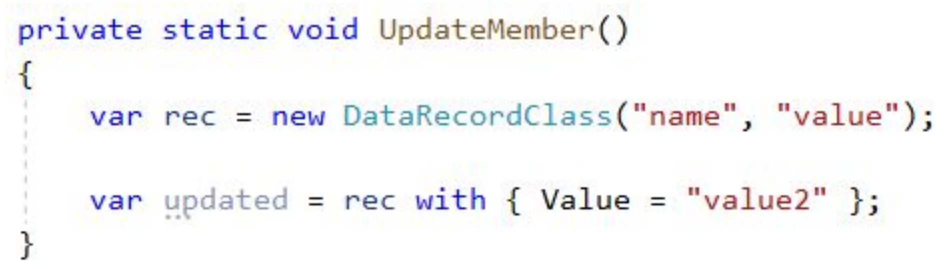
```csharp
private static bool LogExpression(
    bool expression,
    [CallerFilePath] string callerFilePathAttribute = null,
    [CallerLineNumber] int lineNumber = 0,
    [CallerArgumentExpression(nameof(expression))] string expressionText = null)
{
    Console.WriteLine($"{callerFilePathAttribute}:{lineNumber}:{expressionText}={expression}");
    return expression;
}

public static void CheckRange(int val, int upperBound)
{
    if (LogExpression(val < upperBound))
    {
        // do something
    }
}
```



```csharp
private static void PrintDictionary(IDictionary<string, (int X, int Y)> dict)
{
    foreach (var entry in dict)
    {
        Console.WriteLine($"{entry.Value.X}");
    }
}
```



```csharp
private static void PrintDictionary2(IDictionary<string, (int X, int Y)> dict)
{
    foreach (var (_, (X, _)) in dict)
    {
        Console.WriteLine($"{X}");
    }
}
```



```
private static void UpdateMember()
{
    var rec = new DataRecordClass("name", "value");

    var updated = rec with { Value = "value2" };
}
```

https://www.youtube.com/watch?v=O89-zG84QK4

Nullable reference types

- Включается на уровне проекта
- dotnet/runtime и другие проекты уже переведены
- Включено по умолчанию с C# 10 (net 6)