Object Calisthenics

Indentation

Why?

Readability and maintainability

Same number of lines or more, but easiere to understand

```
public static int MaxOneLevelOfIndentationStart()
{
    var total = 0;

    for (var i = 0; i < 5; i++)
    {
        total = AddNumbers(total);
    }

    return total;
}

1 usage
static int AddNumbers(int total)
{
    for (var j = 0; j < 5; j++)
    {
        total += j;
    }

    return total;
}
```

```
public static int MaxOneLevelOfIndentationStart()
{
    var total = 0;

    for (var i = 0; i < 5; i++)
    {
        for (var j = 0; j < 5; j++)
        {
            total += j;
        }
    }

    return total;
}
```

Else

- Why?
- Nested conditionals
- Force you to do more early returns

```
public static bool ElseSucks()
{
    var isSuccess = false;
    if (true)
    {
        isSuccess = true;
    }
    else
    {
        isSuccess = Convert.ToBoolean("POTATO");
    }

    return isSuccess;
}
```

```
public static bool ElseSucks()
{
    if (true)
    {
        return true;
    }

    return Convert.ToBoolean("POTATO");
}
```

Wrap primitives

- Why?
- Readability
- Easier to make changes if used in multiple places

```
public static string WrapPrimitives()
{
    return "Extremely intricate string";
}
```

```
static readonly string ExtremelyIntricateString = "Extremely intricate string";

1 usage
public static string WrapPrimitives()
{
    return ExtremelyIntricateString;
}
```

First-class collections

- Why?
- Centralizes behaviors
- Easier to make unit tests

```
1 usage
public static List<int> FirstClassCollections()
{
    var list = new List<int>();

    list.Add( item: 1);

    return list;
}
```

```
1 usage
public static NumberCollection FirstClassCollections()
{
    var list = new NumberCollection();

    list.Add( item: 1);

    return list;
}
}

2 usages  1 exposing API
public class NumberCollection : IList<int>
{
    readonly IList<int> _listImplementation = new List<int>();
```

One dot per line

- Why?
- Cleaner code

DON’T ABREVIATE

- Why????????
- Abreviations makes the code harder to read
- Why would you want to?

```
public void IHTRNWIVL()
{
    return;
}
```

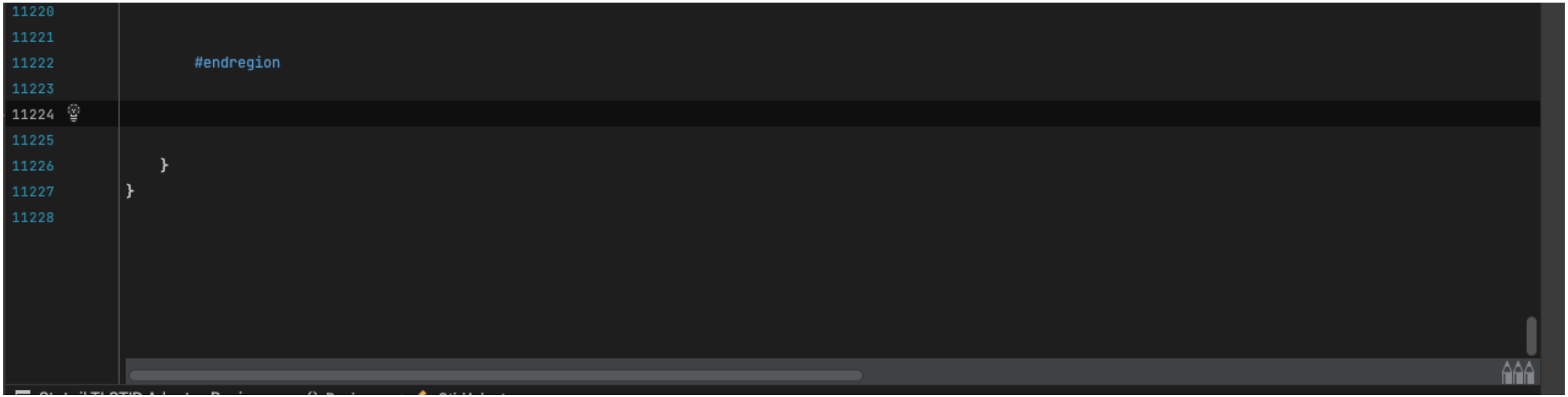
```
public void IncredibleHardToReadNameWhichIsVeryLong()
{
    return;
}
```

```
public int COPO()
{
    return 1 + 1;
}
```

```
public int CalculateOnePlusOne()
{
    return 1 + 1;
}
```

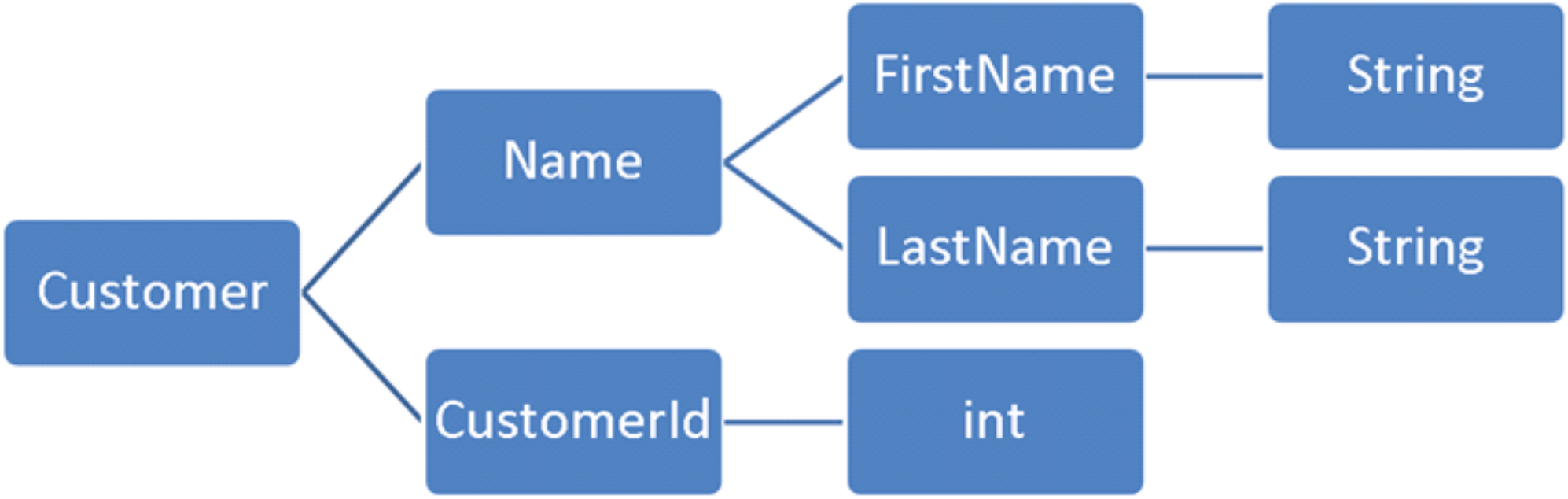
Keep entities small

- [10 files per package, 50 lines per class, 5 lines per method, 2 arguments per method]
- Easier to understand and reuse



No classes with more than two instance variables

- For simplicity
- Hard to actually do



No Getters/Setters/Properties

- Why?
- Keeps logic inside of class
- Each class is responsible for its own data

```
public class Game
{
    2 usages
    public int Score { get; set; }
}
```

```
[Fact]
public void ObjectCalisthenicsNoGettersOrSetters()
{
    var game = new Game();
    game.Score += 5;

    Assert.Equal(expected: 5, actual: game.Score);
}
```

```
public class Game
{
    int _score;

    1 usage
    public void AddScore(int number)
    {
        _score += number;
    }

    1 usage
    public int GetScore()
    {
        return _score;
    }
}
```

```
[Fact]
public void ObjectCalisthenicsNoGettersOrSetters()
{
    var game = new Game();
    game.AddScore(number: 5);

    Assert.Equal(expected: 5, actual: game.GetScore());
}
```

Source

- https://williamdurand.fr/2013/06/03/object-calisthenics/#8-no-classes-with-more-than-two-instance-variables

Thank you ☺