

Object Calisthenics

Some thoughts...



Markus Doggweiler, February 2022, ALCOR Walking

Agenda

My take on Object Calisthenics

- Why?
- How?
- Where and when (not)?



Not-Agenda

What I would like to skip

- A summary of what we have learned already in the lessons
- Origin of name, definition, overview
- A look at the rules in detail

Why should we consider Object Calisthenics?

Because we want simpler code. And simple code is good.

Some reasons for simple code

- Easier to read
- Easier to find bugs/problems
- Easier to refactor
- Easier to reason about

Object Calisthenics help (or force) us to produce simpler code.

How do they help us?

They force us to create smaller things.


There is even one rule that's called "Keep all entities small".

What are "things" or "entities"?

- Classes
- Methods/functions
- Basically any block of code
- Components (UI)

And why is small good?

Easy, obvious, focus, clear

- 
- Because we need to split things up
 - Splitting up forces us to decide what belongs together
 - Every “thing” needs a name – names are good, they give meaning
 - Less clutter
 - Focus on what’s relevant

All of this makes our code (more) obvious.

That’s what we want.

In which context can they be applied?

OOP?

“Object Calisthenics: Principles for Better **Object-Oriented Code**”

[<https://blog.avenuecode.com/object-calisthenics-principles-for-better-object-oriented-code>]

“...that can help you to internalize principles of good **object-oriented design...**”

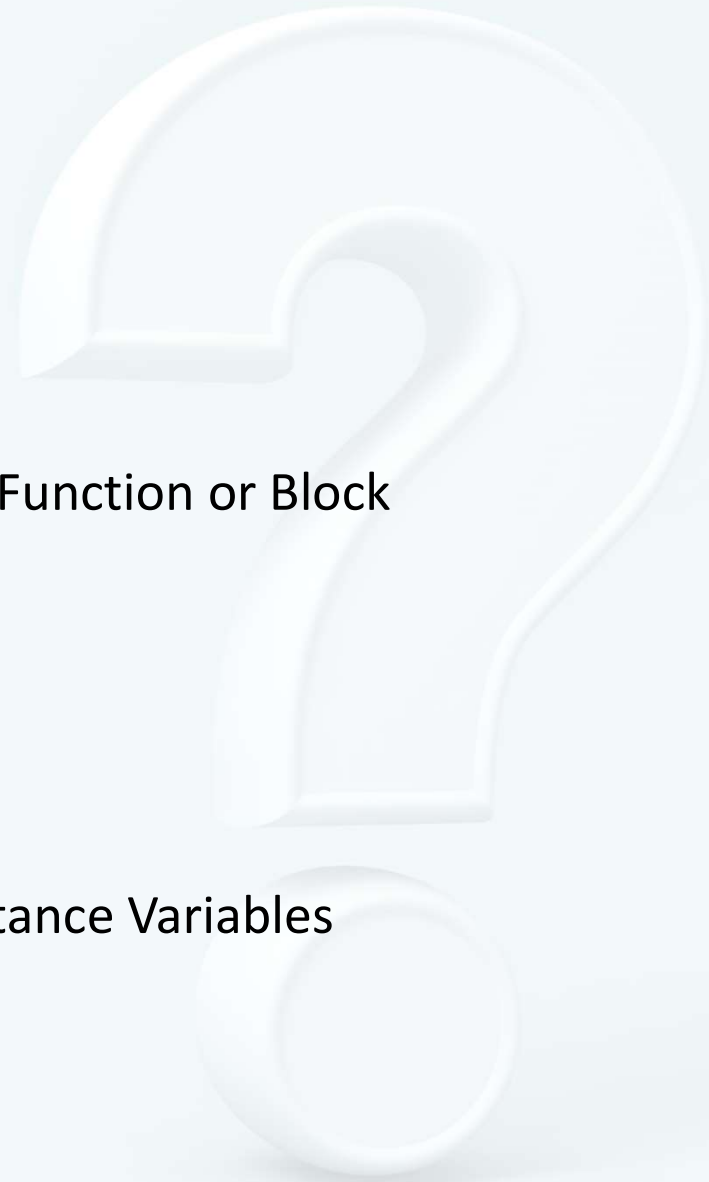
[Jeff Bay, <https://www.bennadel.com/resources/uploads/2012/ObjectCalisthenics.pdf>]

Really? Only OOP?

No!

As soon as there is any code.

- Only One Level Of Indentation Per Method Function or Block
- Don't Use The ELSE Keyword
- Wrap All Primitives And Strings
- One Dot Per Line
- Don't Abbreviate
- Keep All Entities Small
- No ~~Classes~~ Object With More Than Two Instance Variables
- First Class Collections
- No Getters/Setters/Properties



Global ideas...

...that work everywhere *

That's why I like them, they make you a better programmer that writes better code.

Regardless of

- Language
- Framework
- Technology
- (Business) Context

* Disclaimer: "Everywhere" – I'm sure there are exceptions, but there always are... ;)

And maybe the best regarding these rules...

...they can be broken

“By **trying to follow** these rules as much as possible, you will naturally change how you write code. **It doesn't mean you have to** follow all these rules, all the time. Find your balance with these rules, use some of them **only if you feel comfortable** with them.”

[<https://williamdurand.fr/2013/06/03/object-calisthenics/>]

When and how to break?

...no rule without exception(s)...

Rule 1: Only One Level of Indention

```
public String board(String[][] data) {  
    StringBuilder builder = new StringBuilder();  
    int numberOfNonEmptyCells = 0;  
  
    for (int row = 0; row < 10; row++) {  
        for (int column = 0; column < 10; column++) {  
            String value = data[row][column];  
  
            builder.append(value);  
  
            if (value != null) {  
                numberOfNonEmptyCells++;  
            }  
        }  
        builder.append("\n");  
    }  
  
    builder.append("Found " + numberOfNonEmptyCells + " non-empty cells.");  
  
    return builder.toString();  
}
```

```

public String board(String[][] data) {
    StringBuilder builder = new StringBuilder();

    int numberOfNonEmptyCells = processRows(data, builder);

    builder.append("Found " + numberOfNonEmptyCells + " non-empty cells.");

    return builder.toString();
}

```

```

private int processRows(String[][] data, StringBuilder builder) {
    int numberOfNonEmptyCells = 0;

```

Method needs to return number - what does it mean?
Redundant initializations

```

    for (int row = 0; row < 10; row++) {
        numberOfNonEmptyCells += processColumns(builder, data[row]);
        builder.append("\n");
    }

    return numberOfNonEmptyCells;
}

```

```

private int processColumns(StringBuilder builder, String[] rowData) {
    int numberOfNonEmptyCells = 0;

```

Method needs to return number - what does it mean?
Redundant initializations

```

    for (int column = 0; column < 10; column++) {
        numberOfNonEmptyCells += processValue(builder, rowData[column]);
    }

    return numberOfNonEmptyCells;
}

```

```

private int processValue(StringBuilder builder, String value) {
    builder.append(value);

```

Method needs to return number - what does it mean?

```

    if (value != null){
        return 1;
    }

    return 0;
}

```

When and how to break?

...no rule without exception(s)...

Rule 1: Only One Level of Indention

```
public GameState getGameState() {  
    for (WinningCondition condition : winningConditions.getConditions()) {  
        if (fieldsAreEqual(condition.getField1(), condition.getField2(), condition.getField3())) {  
            return getWinnerAt(condition.getField1());  
        }  
    }  
    return GameState.ON_GOING;  
}
```

When and how to break?

...no rule without exception(s)...

Rule 6: Don't Abbreviate

- Very well-known abbreviations (e.g. SP for SharePoint)
- i as index in loops
- Parameter name in lambdas, arrow functions, etc.

```
_measurements  
    .OrderBy(m => m.DateTime)  
    .Where(m => m.Value > 10)  
    .ToArray();
```

Takeaways

What we talked about...

- Use Object Calisthenics often, ...
- ... when you feel comfortable ...
- ... and/or they make sense.

- Try to keep things small,
- with good names (= a clear purpose),
- make it obvious!

Have fun!



Thanks for your attention!

Any Questions?

contact me later at

markus.doggweiler@gmail.com

<https://www.linkedin.com/in/markus-doggweiler-745817/>